Adventures of Two Little Owls in Rule Land

Markus Krotzsch

University of Oxford Computing Laboratory
OWL & Rules

- Knowledge Representation on the Semantic Web: OWL 2
  - W3C standard for Web Ontology Language
  - Related description logic: SROIQ

- Many approaches for combining OWL/DL and “rules” were proposed:
  - SWRL, DLP, CARIN, AL-Log, dl-programs, DL-safe rules, DL Rules, …
  - Reconciling both worlds is difficult

- OWL 2 Profiles:
  - Light-weight OWL fragments
  - Easier to combine with rule-like approaches
The Owl Profiles
OWL Lite as failure:

- Defined as fragment of OWL 1 DL, intended to be simpler
- However: almost as complex as OWL DL (ExpTime)
- Complex syntax hides real expressive power
- Current usage in ontologies coincidental rather than intentional

Original goal: simpler implementation and usage

→ approach in OWL 2: three simpler **language profiles**:

- OWL 2 QL
- OWL 2 EL
- OWL 2 RL
OWL 2 Profiles

Design principle for profiles:
Identify maximal OWL 2 sublanguages that are still implementable in PTime.

Main source of intractability: non-determinism (requires guessing/backtracking)
- owl:unionOf, or owl:complementOf + owl:intersectionOf
- Max. cardinality restrictions
- Combining existentials (owl:someValuesFrom) and universals (owl:allValuesFrom) in superclasses
- Non-unary finite class expressions (owl:oneOf) or datatype expressions

→ features that are not allowed in any OWL 2 profile
### Overview: Essential OWL Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Related OWL vocabulary</th>
<th>FOL</th>
<th>DL</th>
</tr>
</thead>
<tbody>
<tr>
<td>top/bottom class</td>
<td>owl:Thing/owl:Nothing</td>
<td>(axiomatise)</td>
<td>⊤/⊥</td>
</tr>
<tr>
<td>Class intersection</td>
<td>owl:intersectionOf</td>
<td>∧</td>
<td>⊓</td>
</tr>
<tr>
<td>Class union</td>
<td>owl:unionOf</td>
<td>∨</td>
<td>⊔</td>
</tr>
<tr>
<td>Class complement</td>
<td>owl:complementOf</td>
<td>¬</td>
<td>¬</td>
</tr>
<tr>
<td>Enumerated class</td>
<td>owl:oneOf</td>
<td>(ax. with ≈)</td>
<td>{a}</td>
</tr>
<tr>
<td><strong>Property restrictions</strong></td>
<td><strong>owl:onProperty</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Existential</td>
<td>owl:someValueFrom</td>
<td>∃y …</td>
<td>∃R.C</td>
</tr>
<tr>
<td>Universal</td>
<td>owl:allValuesFrom</td>
<td>∀y …</td>
<td>∀R.C</td>
</tr>
<tr>
<td>Min. cardinality</td>
<td>owl:minQualifiedCardinality/owl:onClass</td>
<td>∃y1…yn….</td>
<td>≥n S.C</td>
</tr>
<tr>
<td>Max. cardinality</td>
<td>owl:maxQualifiedCardinality/owl:onClass</td>
<td>∀y1…yn+1. … → …</td>
<td>≤n S.C</td>
</tr>
<tr>
<td>Local reflexivity</td>
<td>owl:hasSelf</td>
<td>R(x,x)</td>
<td>∃R.Self</td>
</tr>
</tbody>
</table>

Markus Krötzsch: Adventures of Two Little OWLs in Rule Land
## Overview: Essential OWL Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Related OWL vocabulary</th>
<th>DL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property chain</td>
<td>owl:propertyChainAxiom</td>
<td></td>
</tr>
<tr>
<td>Inverse</td>
<td>owl:inverseOf</td>
<td>R⁻</td>
</tr>
<tr>
<td>Key</td>
<td>owl:hasKey</td>
<td>Actually a rule</td>
</tr>
<tr>
<td>Property disjointness</td>
<td>owl:propertyDisjointWith</td>
<td>Dis(R,S)</td>
</tr>
</tbody>
</table>

**Property characteristics**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Related OWL vocabulary</th>
<th>DL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric</td>
<td>owl:SymmetricProperty</td>
<td>Sym(R)</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>owl:AsymmetricProperty</td>
<td>Asy(R)</td>
</tr>
<tr>
<td>Reflexive</td>
<td>owl:ReflexiveProperty</td>
<td>Ref(R)</td>
</tr>
<tr>
<td>Irreflexive</td>
<td>owl:IrreflexiveProperty</td>
<td>Irr(R)</td>
</tr>
<tr>
<td>Transitivity</td>
<td>owl:TransitiveProperty</td>
<td>Tra(R)</td>
</tr>
</tbody>
</table>

| Subclass                       | rdfs:subClassOf                                  | ∀x.C(x) → D(x) | CED |
| Subproperty                    | rdfs:subPropertyOf                               | ∀x,y.R(x,y) → S(x,y) | RES |
OWL 2 EL

OWL profile based on description logic EL++

- Intuition: focus on terminological expressivity used for light-weight ontologies
- Allow `owl:someValuesFrom` (existential) but not `owl:allValuesFrom` (universal)
- Property domains, class/property hierarchies, class intersections, disjoint classes/properties, property chains, `owl:hasSelf`, `owl:hasValue`, and keys fully supported
- No inverse or symmetric properties
- `rdfs:range` allowed but with some restrictions
- No `owl:unionOf` or `owl:complementOf`
- Various restrictions on available datatypes
OWL 2 EL: Features

- Standard reasoning in OWL 2 EL: PTime-complete
- Used by practically relevant ontologies: Prime example is SNOMED CT (clinical terms ontology with classes and properties in the order of $10^5$)
- Fast implementations available: full classification of SNOMED-CT in <1min; real-time responsivity when preprocessed (modules)
OWL 2 QL

OWL profile that can be used to query data-rich applications:

- Intuition: use OWL concepts as light-weight queries, allow query answering using rewriting in SQL on top of relational DBs
- Different restrictions on subclasses and superclasses of `rdfs:SubclassOf`:
  - subclasses can only be class names or `owl:someValuesFrom` (existential) with unrestricted (`owl:Thing`) filler
  - superclasses can be class names, `owl:someValuesFrom` or `owl:intersectionOf` with superclass filler (recursive), or `owl:complementOf` with subclass filler
- Property hierarchies, disjointness, inverses, (a)symmetry supported, restrictions on range and domain
- Disjoint or equivalence of classes only for subclass-type expressions
- No `owl:unionOf`, `owl:allValuesFrom`, `owl:hasSelf`, `owl:hasKey`, `owl:hasValue`, `owl:oneOf`, `owl:sameAs`, `owl:propertyChainAxiom`, `owl:TransitiveProperty`, cardinalities, functional properties
- Some restrictions on available datatypes
OWL 2 QL: Features

- Standard reasoning in OWL 2 QL: PTime, for some cases even <PTime
- Convenient light-weight interface to legacy data
- Fast implementations on top of legacy database systems (relational or RDF): highly scalable to very large datasets
OWL 2 RL

OWL profile that resembles an OWL-based rule language:

- Intuition: subclass axioms in OWL RL can be understood as rule-like implications with head (superclass) and body (subclass)
- Different restrictions on subclasses and superclasses of `rdfs:SubclassOf`:
  - subclasses can only be class names, `owl:oneOf`, `owl:hasValue`, `owl:intersectionOf`, `owl:unionOf`, `owl:someValuesFrom` if applied only to subclass-type expressions
  - superclasses can be class names, `owl:allValuesFrom` or `owl:hasValue`; also max. cardinalities of 0 or 1 are allowed, all with superclass-type filler expressions only
- Property domains and ranges only for subclass-type expressions; property hierarchies, disjointness, inverses, (a)symmetry, transitivity, chains, (inverse)functionality, irreflexivity fully supported
- Disjoint classes and classes in keys need subclass-type expressions, equivalence only for expressions that are sub- and superclass-type, no restrictions on `owl:sameAs`
- Some restrictions on available datatypes
OWL 2 RL: Features

- Standard reasoning in OWL 2 RL: PTime-complete
- Rule-based reading simplifies modelling and implementation: even naïve implementations can be useful
- Fast and scalable implementations exist
Do We Really Need So Many OWLs?

Three new OWL profiles with somewhat complex descriptions … why not just one?

- The union of any two of the profiles is no longer light-weight! QL+RL, QL+EL, RL+EL all ExpTime-hard
- Restricting to fewer profiles = giving up potentially useful feature combinations
- Rationale: profiles are “maximal” (well, not quite) well-behaved OWL 2 fragments → Pick suitable feature set for applications
- In particular, nobody is forced to implement all of a profile
Entering Rule Land: Instance Retrieval with Rule Systems
Translating OWL RL to Rules

- OWL RL builds on the “Description Logic Programming” idea → all axioms can be written as Horn logic rules

- Examples:
  - $C \sqsubseteq D$  
    \[ C(x) \rightarrow D(x) \]
  - $C \sqcap D \sqsubseteq E$  
    \[ C(x) \land D(x) \rightarrow E(x) \]
  - $C \sqsubseteq \forall R.D$  
    \[ C(x) \land R(x,y) \rightarrow D(y) \]
  - $\exists R.D \sqsubseteq E$  
    \[ R(x,y) \land D(y) \rightarrow E(x) \]
  - $C \sqsubseteq \exists R.\{a\}$  
    \[ C(x) \rightarrow R(x,a) \]

- One axiom – one rule
Translating OWL RL to Facts

- Rule engines work well with many facts but few rules → transform OWL RL to facts, processed by meta-rules

- Examples:
  - $C \subseteq D$ \quad SubClass(C,D)
  - $C \cap D \subseteq E$ \quad SubConj(C,D,E)
  - $C \subseteq \forall R.D$ \quad SupForall(C,R,D)
  - $\exists R.D \subseteq E$ \quad SubEx(R,D,E)
  - $C \subseteq \exists R.\{a\}$ \quad SupExOne(C,R,a)

- One axiom – some facts
Meta-Rules for OWL RL Reasoning

- Relevant relationships can be modelled in static rules:

\[
\text{inst}(x,y) \land \text{SubClass}(y,z) \rightarrow \text{inst}(x,z) \\
\text{inst}(x,y_1) \land \text{inst}(x,y_2) \land \text{SubConj}(y_1,y_2,z) \rightarrow \text{inst}(x,z)
\]

... (and many more, see [M+2009] in the reference section)

- OWL 2 Specification provides complete rule set for OWL RL
  - No translation to datalog facts
  - Rules directly expressed on RDF graphs
  - (but otherwise it's the same, really)
A Simple EL-type Description Logic: $ELO$

- **Building blocks:**
  - **individuals $I$, concepts $A$** (unary pred.), **roles $R$** (binary pred.)

- **Concept expressions of $ELO$:**

  $$ C ::= A \mid \{I\} \mid (C \sqcap C) \mid \exists R.C $$

- **Axioms of $ELO$:**

  - $C1 \sqsubseteq C2$ (concept inclusion)
  - $C(a)$ (concept assertion, $\{a\} \sqsubseteq C$)
  - $R(a,b)$ (role assertion, $\{a\} \sqsubseteq \exists R.\{b\}$)
Translation to Datalog (with some Twists)

- Example:

\[
\begin{align*}
C(a) & \quad C \sqsubseteq D & \quad C \cap D \sqsubseteq E
\end{align*}
\]

- Write DL axioms as datalog facts:

\[
\begin{align*}
\text{SubClass}(a,C) & \quad \text{SubClass}(C,D) & \quad \text{SubConj}(C,D,E) \\
\text{Nom}(a)
\end{align*}
\]

- Relevant deduction rules:

\[
\begin{align*}
\text{Nom}(x) \rightarrow \text{inst}(x,x) \\
\text{SubClass}(y,z) \land \text{inst}(x,y) \rightarrow \text{inst}(x,z) \\
\text{SubConj}(y_1,y_2,z) \land \text{inst}(x,y_1) \land \text{inst}(x,y_2) \rightarrow \text{inst}(x,z)
\end{align*}
\]
Dealing with Existential Quantifiers

- Example:

\[ \{a\} \sqsubseteq \exists R. C \]  
\[ C \sqsubseteq D \]  
\[ \exists R. D \sqsubseteq E \]

- Solution: introduce auxiliary constants in datalog

\[ \text{SupEx}(a,R,C,aux) \]  
\[ \text{SubClass}(C,D) \]  
\[ \text{SubEx}(R,D,E) \]  
\[ \text{Nom}(a) \]

- Additional deduction rules:

\[ \text{SupEx}(y,v,z,x') \rightarrow \text{inst}(x',z) \]  
\[ \text{SupEx}(y,v,z,x') \land \text{SubEx}(v,y',z') \land \text{inst}(x,y) \land \text{inst}(x',y') \rightarrow \text{inst}(x,z') \]
An Instance Retrieval Calculus for \textit{ELO}

Rules for a sound & complete instance retrieval calculus for \textit{ELO}:

\begin{align*}
\text{Nom}(x) & \rightarrow \text{inst}(x,x) \\
\text{SubClass}(y,z) \land \text{inst}(x,y) & \rightarrow \text{inst}(x,z) \\
\text{SubConj}(y_1,y_2,z) \land \text{inst}(x,y_1) \land \text{inst}(x,y_2) & \rightarrow \text{inst}(x,z) \\
\text{SupEx}(y,v,z,x') & \rightarrow \text{inst}(x',z) \\
\text{SupEx}(y,v,z,x') \land \text{SubEx}(v,y',z') \land \text{inst}(x,y) \land \text{inst}(x',y') & \rightarrow \text{inst}(x,z') \\
\text{inst}(x,y) \land \text{Nom}(y) \land \text{inst}(x,z) & \rightarrow \text{inst}(y,z) \\
\text{inst}(x,y) \land \text{Nom}(y) \land \text{inst}(y,z) & \rightarrow \text{inst}(x,z)
\end{align*}

(A complete rule set for OWL EL if found in \cite{K.2010}, see references)
Comparison of OWL RL and EL Retrieval

**OWL RL**
- Ontologies as data
- Reasoning encoded in rules
- Typically forward-chained (materialisation)
- Rules can naturally be written in OWL/RDF syntax, translation can be omitted
- Rules under first-order semantics equivalent to OWL RL under Direct Semantics

**OWL EL**
- Ontologies as data
- Reasoning encoded in rules
- Typically forward-chained (materialisation)
- Rules require (a few) auxiliary constants, translation needed
- Rules under first-order semantics not semantically equivalent to OWL EL
Reforging the Owl that was Broken: Combining Owl RL & EL (and some rules)
Rules as a Base for Integrating EL and RL

- OWL EL + OWL RL:
  - All standard reasoning tasks 2ExpTime-complete

- Rule encoding of EL + rule encoding of RL
  - Fixed set of rules: reasoning PTime-complete (datalog data compl.)

A meaningful approximation of EL + RL reasoning?
Rules as a Base for Integrating EL and RL

- OWL EL + OWL RL:
  - All standard reasoning tasks 2ExpTime-complete

- Rule encoding of EL + rule encoding of RL
  - Fixed set of rules: reasoning PTime-complete (datalog data compl.)

A meaningful approximation of EL + RL reasoning?

No!

EL existential encoding not valid in presence of RL axioms
unsound conclusions
Ensuring Soundness: DL-safety

- **Problem:** “Skolem constants” introduced for OWL EL cannot be treated like normal constants when applying OWL RL rules.

- **Solution:** Introduce a “guard” predicate to prevent this.
  - For all genuine DL individual names, add fact O(x)
  - Restrict all OWL RL meta rules to apply only to elements in O

- **Intuition:**
  - OWL RL rules apply only to ABox facts
  - Interactions of OWL EL and OWL RL on existentially implied elements is ignored
Adding Further Rules

OWL RL translation (ABox) semantically faithful → Further DL-safe rules can be added for modelling

Resulting unified rule-based reasoning system:
- Sound
- Complete for EL-only input
- Complete for RL-only input, even with added rules
- Incomplete for EL+RL or EL+Rules
- Fixed rule set for EL+RL: polynomial time algorithm
- Dynamic set of additional (DL-safe) rules: ExpTime compete

- Note: DL-safety can be built into the rule engine/reasoner instead of adding an O predicate
FROM INSTANCE RETRIEVAL TO CLASSIFICATION
From Instance Retrieval to Classification

- The previous algorithms only compute instance-of relations (Corollary: W3C OWL RL rules are not sufficient for solving all ontology entailment tasks.)

- How to check $A \sqsubseteq B$ using an algorithm for instance retrieval?
  - Assume that $A(a)$ is given
  - Check whether $B(a)$ follows
From Instance Retrieval to Classification

- The previous algorithms only compute instance-of relations (Corollary: W3C OWL RL rules are not sufficient for solving all ontology entailment tasks.)

- How to check $A \sqsubseteq B$ using an algorithm for instance retrieval?
  - Assume that $A(a)$ is given
  - Check whether $B(a)$ follows

- This just checks for one subsumption, but assuming $A(a)$ may lead to other entailments – cannot do all checks together.

→ No algorithm for materialising all subsumptions yet!
From Instance Retrieval to Classification

- Can we internalise individual tests into the rule set?
- Idea: Record test assumption that led to an inference:
  
  \[ \text{inst}(x,y) \quad \text{becomes} \quad \text{inst}(x,y,A) \]

- We (ab)use class names as “test instances” in datalog, use \text{inst()} to compute subclass relationships
- All tests can be executed in parallel, using forward chaining
“inst(C,D,C)” now encodes “C is a subclass of D”

\[
\begin{align*}
\text{Class}(q) & \rightarrow \text{inst}(q,q,q) \\
\text{Nom}(x) \land \text{Class}(q) & \rightarrow \text{inst}(x,x,q) \\
\text{SubClass}(y,z) \land \text{inst}(x,y,q) & \rightarrow \text{inst}(x,z,q) \\
\text{SubConj}(y_1,y_2,z) \land \text{inst}(x,y_1,q) \land \text{inst}(x,y_2,q) & \rightarrow \text{inst}(x,z,q) \\
\text{SupEx}(y,v,z,x') \land \text{Class}(q) & \rightarrow \text{inst}(x',z,q) \\
\text{SupEx}(y,v,z,x') \land \text{SubEx}(v,y',z') \land \text{inst}(x,y,q) \land \text{inst}(x',y',q) & \rightarrow \text{inst}(x,z',q) \\
\text{inst}(x,y,q) \land \text{Nom}(y) \land \text{inst}(x,z,q) & \rightarrow \text{inst}(y,z,q) \\
\text{inst}(x,y,q) \land \text{Nom}(y) \land \text{inst}(y,z,q) & \rightarrow \text{inst}(x,z,q)
\end{align*}
\]

Problem: many redundant inferences, higher space requirements
Could Classification be more Efficient?

- Do we really need this additional class parameter?
Could Classification be more Efficient?

- Do we really need this additional class parameter?
- A known “binary” classification calculus for $\mathcal{EL}$:

\[
\begin{align*}
\text{Class}(q) & \rightarrow \text{inst}(q,q) \\
\text{Nom}(x) & \rightarrow \text{inst}(x,x) \\
\text{SubClass}(y,z) & \land \text{inst}(x,y) \rightarrow \text{inst}(x,z) \\
\text{SubConj}(y_1,y_2,z) & \land \text{inst}(x,y_1) \land \text{inst}(x,y_2) \rightarrow \text{inst}(x,z) \\
\text{SupEx}(y,v,z,x') & \rightarrow \text{inst}(x',z) \\
\text{SupEx}(y,v,z,x') & \land \text{SubEx}(v,y',z') \land \text{inst}(x,y) \land \text{inst}(x',y') \rightarrow \text{inst}(x,z')
\end{align*}
\]
A Binary Classification Calculus for $\mathcal{ELO}$?

- The problem with nominals:

  \[
  A \subseteq \exists R_1.C_1 \quad A \subseteq \exists R_2.C_2 \quad \exists R_1.C_2 \subseteq E \quad C_1 \subseteq \{b\} \quad C_2 \subseteq \{b\}
  \]

  \[
  \rightarrow \quad C_1 \subseteq C_2 \text{ follows if } A \text{ is non-empty.}
  \]
A Binary Classification Calculus for $\mathcal{ELO}$?

- The problem with nominals:

  \[ A \sqsubseteq \exists R_1. C_1 \quad A \sqsubseteq \exists R_2. C_2 \quad \exists R_1. C_2 \sqsubseteq E \quad C_1 \sqsubseteq \{b\} \quad C_2 \sqsubseteq \{b\} \]

  \[ \rightarrow \quad C_1 \sqsubseteq C_2 \text{ follows if } A \text{ is non-empty.} \]

- We could cover this case with a new rule …

  … but would this be enough?
A Binary Classification Calculus for $\mathcal{ELO}$?

- The problem with nominals:

$$A \sqsubseteq \exists R_1.C_1 \quad A \sqsubseteq \exists R_2.C_2 \quad \exists R_1.C_2 \sqsubseteq E \quad C_1 \sqsubseteq \{b\} \quad C_2 \sqsubseteq \{b\}$$

$$\rightarrow C_1 \sqsubseteq C_2 \text{ follows if } A \text{ is non-empty.}$$

- We could cover this case with a new rule …

… but would this be enough?

No!
Proof Sketch

- **Claim:** If a classification calculus is sound and complete for $ELO$, then it has some inferred predicates of arity 3 or more.

- **Proof by contradiction:** any complete binary calculus has some derivation from which we can construct a wrong derivation:
  1) Suppose there was an arity 2 calculus.
  2) Find a knowledge base that has an interesting entailment. Then the calculus must find this entailment.
  3) Transform this proof tree into another valid proof tree for another input.
  4) Show that this proof tree leads to a wrong entailment.
Transforming Proof Trees

SubConj(A, B, C)

inst(A, C)

inst(A, A)

inst(A, B)

Class(A)

SubClass(A, B)

inst(A, A)

Class(A)
Transforming Proof Trees

- **Idea:** Rename the symbols that are not used further up in the tree
  → Rename symbols below a node that do not appear in the node's label
Transforming Proof Trees

- **Idea:** Rename the symbols that are not used further up in the tree
  → Rename symbols below a node that do not appear in the node's label
  → **proof tree for modified input:** Class(A)  SubConj(A, B', C)  SubClass(A, B')

- Symbols are renamed below each tree node:
symbols of **at most 2 input axioms** are shared with the rest of the tree
An Interesting Knowledge Base

- **KB (for some k>0):**
  
  For $i=0,\ldots,k$:
  
  $D_i \sqsubseteq \exists S_i.D_{i+1}$  
  $\exists S_i.B_{i+1} \sqsubseteq B_i$
  
  $D_0 \sqsubseteq \exists R.A$  
  $A \sqsubseteq B_{k+1}$
  
  $D_{k+1} \sqsubseteq \{a\}$  
  $A \sqsubseteq \{a\}$

  $\rightarrow$ KB entails $D_0 \sqsubseteq B_0$

- **Dependency graph:**

  ![Dependency graph diagram](image)
Finishing the Proof

- Take a KB proof tree for \( D_0 \sqsubseteq B_0 \) — transform it — get a modified KB'

- KB' can entail \( D_0 \sqsubseteq B_0 \) only if KB' still has a dependency graph like this →

\[ \begin{align*}
\text{Sub(proof)tree} &= \text{Sub-knowledge-base} = \text{Sub(dependency)graph} \\
\text{Symbols shared with rest of the tree} &= \text{Symbols shared with axioms not in sub-KB} = \text{Nodes shared with rest of dependency graph}
\end{align*} \]

- **Observation**: If a subgraph contains between 3 and \( k-3 \) axioms of the form \( D_i \sqsubseteq \exists S_i. D_{i+1} \), then it touches the rest of the graph in at least three axioms.
Conclusions and Results
Further Results on EL-type Logics [K.2010]

- Overview of results:

<table>
<thead>
<tr>
<th></th>
<th>ELO</th>
<th>OWL EL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL supports role chains?</td>
<td>–</td>
<td>yes</td>
</tr>
<tr>
<td>DL supports nominals?</td>
<td>–</td>
<td>yes</td>
</tr>
<tr>
<td>Minimal arity for instance retrieval</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Minimal arity for classification</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- What this tells us:
  - Classification is harder than instance retrieval
  - Some DLs are more polynomial than others ;-
  - Problematic features: role chains, nominals, universal (top) roles
  - Non-problematic features: Self, role conjunctions, range restrictions
Further Results on RL-type Logics

- Conjecture: all results carry over (existentials not crucial)

<table>
<thead>
<tr>
<th></th>
<th>–</th>
<th>yes</th>
<th>–</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL supports role chains?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL supports nominals?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimal arity for instance retrieval</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Minimal arity for classification</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- What this tells us:
  - Classification is harder than instance retrieval
  - Some DLs are more polynomial than others ;-
  - Problematic features: role chains, nominals, universal (top) roles
  - Non-problematic features: Self, role conjunctions, range restrictions
Further Practical Implications

- RDF is too weak a language for describing rule-based calculi.
  - Relevant problems require more than just triples (ternary predicates)
  - One could use blank nodes in rule heads as a workaround, with all complications that this brings (materialisation termination criterion)

- Instance retrieval is easier than classification.
  - Other terminological reasoning tasks are usually as hard as classification

- Choosing an OWL profile is not the most important decision for implementors.
  - EL and RL suggest very similar implementation techniques
  - Excluding single features from these profiles may have a bigger impact
...to Rule them All
References