PRACTICAL USES OF EXISTENTIAL RULES IN KNOWLEDGE REPRESENTATION

Part 2: Existential Rules in Knowledge Representation

David Carral,[1] **Markus Krötzsch,**[1] **and Jacopo Urbani**[2]
1. TU Dresden
2. Vrije Universiteit Amsterdam

Special thanks to Irina Dragoste,[1] **Ceriel Jacobs,**[2] **and Maximilian Marx**[1]
for their invaluable contributions to the software used in this tutorial

ECAI, 4 September 2020

Motivation

“Rules” are the epitome of symbolic reasoning:

- Many logical theories can be represented as rules
- Rules of inference are used to define deduction procedures

$\leadsto$ knowledge representation & reasoning as natural application area for existential rules

Goals for this part:

- Explain how to use rules to solve (quite unrelated) KRR problems
- Illustrate some useful modelling techniques
- Discuss aspects of reasoning performance

Description Logics

Description logics (DLs) are influential and widely used ontology languages

- basis of the W3C Web Ontology Language standard OWL
- specific DLs achieve good trade-offs between expressivity and complexity

Schema modelling in DLs: DLs talk about relational models that use only

- classes (unary predicates), e.g., “drink”
- properties (binary predicates), e.g., “madeWith”

DL ontologies describe relationships between these entities, such as

- subclass relations, e.g.,
  limeSyrup $\sqsubseteq$ fruitSyrup states that “every lime syrup is also a fruit syrup”
- subproperty relations, e.g.,
  madeWith $\sqsubseteq$ contains states that “if x is made with y, then x contains y”

$\rightsquigarrow$ DLs can model general terminological knowledge independent of specific facts

The DL $\mathcal{EL}_\bot^+$ in a nutshell

The $\mathcal{EL}$ family of DLs is simple and supports polynomial time standard reasoning

The DL $\mathcal{EL}_\bot^+$ supports the following class expressions to describe derived classes:

$\bot$	empty class (bottom)	“the empty set”
$\top$	universal class (top)	“set of all elements”
$\exists R.C$	existential restriction	“set of all elements that have an R-relation to some element in class C”
$C \sqcap D$	intersection	“set of all elements that are in class C and in class D”

The DL $\mathcal{EL}^{+}_{\perp}$ in a nutshell

The $\mathcal{EL}$ family of DLs is simple and supports polynomial time standard reasoning

The DL $\mathcal{EL}^{+}_{\perp}$ supports the following class expressions to describe derived classes:

$\perp$	empty class (bottom)	"the empty set"
$\top$	universal class (top)	"set of all elements"
$\exists R.C$	existential restriction	"set of all elements that have an R-relation to some element in class C"
$C \sqcap D$	intersection	"set of all elements that are in class C and in class D"

Class expressions and properties can be used in axioms:

$C \sqsubseteq D$	class subsumption	"Every C is also a D"
$R \sqsubseteq S$	property subsumption	"Every relation of type R is also one of type S"
$R \circ S \sqsubseteq T$	property chain	"Elements connected by a chain of relations R followed by S are also directly connected by T"

$\mathcal{EL}^{+}_{\perp}$ and existential rules

All axioms of $\mathcal{EL}^{+}_{\perp}$ can be rewritten as existential rules

Example: The axiom

$$\mathsf{alcoholicBeverage} \sqsubseteq \mathsf{Drink} \sqcap \exists \mathsf{contains}.\mathsf{Alcohol}$$

can be written as a rule

$$\mathsf{alcoholicBeverage}(x) \to \exists y.\mathsf{Drink}(x) \wedge \mathsf{contains}(x, y) \wedge \mathsf{Alcohol}(y)$$

$\mathcal{EL}_\bot^+$ and existential rules

All axioms of $\mathcal{EL}_\bot^+$ can be rewritten as existential rules

Example: The axiom

$$\mathsf{alcoholicBeverage} \sqsubseteq \mathsf{Drink} \sqcap \exists \mathsf{contains}.\mathsf{Alcohol}$$

can be written as a rule

$$\mathsf{alcoholicBeverage}(x) \to \exists y.\mathsf{Drink}(x) \wedge \mathsf{contains}(x, y) \wedge \mathsf{Alcohol}(y)$$

In general: this works for all Horn Description Logics

Problem: DLs are based on different reasoning methods. The rules they yield do often not lead to a terminating chase.

Reasoning for DLs

Example: A small $\mathcal{EL}_\bot^+$ ontology about drinks:

$$\begin{aligned}
\mathsf{Highball} &\sqsubseteq \mathsf{Drink} \sqcap \exists \mathsf{madeWith}.\mathsf{Spirit} \\
\mathsf{Spirit} &\sqsubseteq \exists \mathsf{contains}.\mathsf{Alcohol} \\
\mathsf{Drink} \sqcap \exists \mathsf{contains}.\mathsf{Alcohol} &\sqsubseteq \mathsf{alcoholicBeverage} \\
\mathsf{madeWith} \circ \mathsf{contains} &\sqsubseteq \mathsf{contains}
\end{aligned}$$

From this example, we should be able to conclude $\mathsf{Highball} \sqsubseteq \mathsf{alcoholicBeverage}$.

Definition: The task of computing all logically entailed subsumptions $A \sqsubseteq B$ between atomic classes A and B is called classification.

Classification for $\mathcal{EL}_\bot^+$ is polynomial, but how exactly should we compute it in rules?

Prior research . . .

Published: 17 November 2013

The Incredible ELK

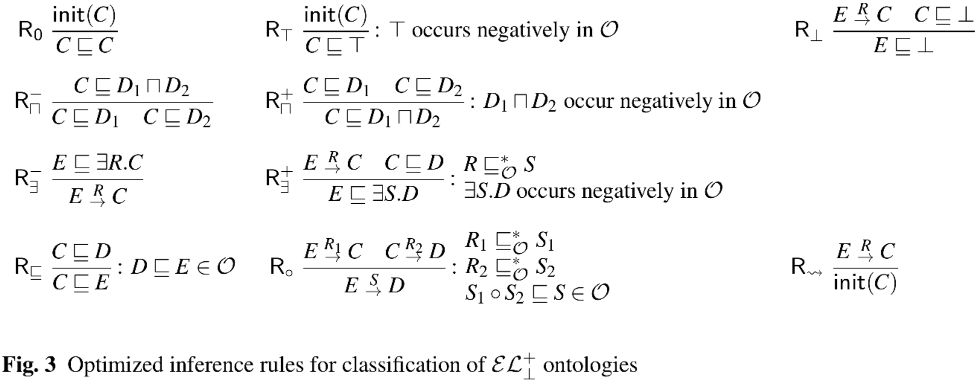
From Polynomial Procedures to Efficient Reasoning with $\mathcal{EL}$ Ontologies

Yevgeny Kazakov, Markus Krötzsch & František Simančík

Journal of Automated Reasoning **53**, 1–61(2014) | Cite this article

518 Accesses | **93** Citations | Metrics

Prior research . . .



$$\mathsf{R}_0\ \frac{\mathsf{init}(C)}{C \sqsubseteq C} \qquad \mathsf{R}_\top\ \frac{\mathsf{init}(C)}{C \sqsubseteq \top} : \top \text{ occurs negatively in } \mathcal{O} \qquad \mathsf{R}_\bot\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq \bot}{E \sqsubseteq \bot}$$

$$\mathsf{R}^-_\sqcap\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad \mathsf{R}^+_\sqcap\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occur negatively in } \mathcal{O}$$

$$\mathsf{R}^-_\exists\ \frac{E \sqsubseteq \exists R.C}{E \xrightarrow{R} C} \qquad \mathsf{R}^+_\exists\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D} : \begin{array}{l} R \sqsubseteq^*_\mathcal{O} S \\ \exists S.D \text{ occurs negatively in } \mathcal{O} \end{array}$$

$$\mathsf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathsf{R}_\circ\ \frac{E \xrightarrow{R_1} C \quad C \xrightarrow{R_2} D}{E \xrightarrow{S} D} : \begin{array}{l} R_1 \sqsubseteq^*_\mathcal{O} S_1 \\ R_2 \sqsubseteq^*_\mathcal{O} S_2 \\ S_1 \circ S_2 \sqsubseteq S \in \mathcal{O} \end{array} \qquad \mathsf{R}_\leadsto\ \frac{E \xrightarrow{R} C}{\mathsf{init}(C)}$$

Fig. 3 Optimized inference rules for classification of $\mathcal{EL}^+_\bot$ ontologies

How to read such rules

General form of the rules:

$$\text{rule name} \quad \frac{\text{pre-condition}}{\text{conclusion}} : \text{side condition}$$

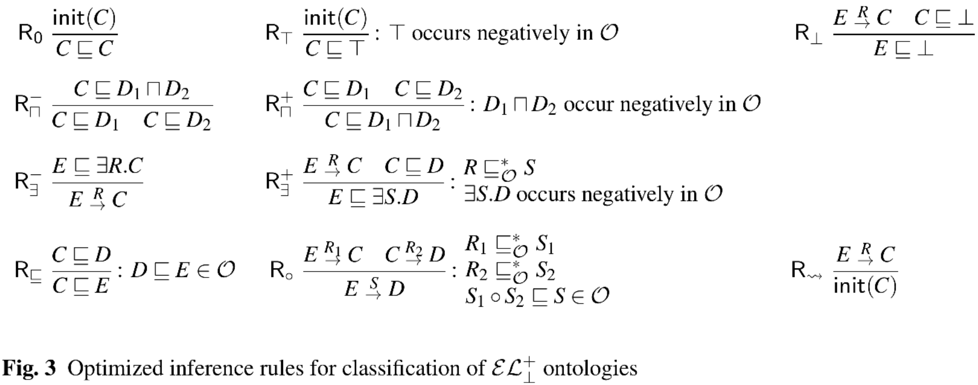
For example:

$$\mathsf{R}^{+}_{\sqcap} \ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occur negatively in } \mathcal{O}$$

where the parts have the following meaning:

- $\mathcal{O}$: the given $\mathcal{EL}^{+}_{\bot}$ ontology
- C, D_1, D_2: arbitrary (possibly nested) $\mathcal{EL}^{+}_{\bot}$ class expressions
- "to occur negatively": to appear in a subclass position

Encoding a calculus in rules



$$\mathsf{R}_0\ \frac{\mathsf{init}(C)}{C \sqsubseteq C} \qquad \mathsf{R}_\top\ \frac{\mathsf{init}(C)}{C \sqsubseteq \top} : \top \text{ occurs negatively in } \mathcal{O} \qquad \mathsf{R}_\bot\ \frac{E \overset{R}{\rightarrow} C \quad C \sqsubseteq \bot}{E \sqsubseteq \bot}$$

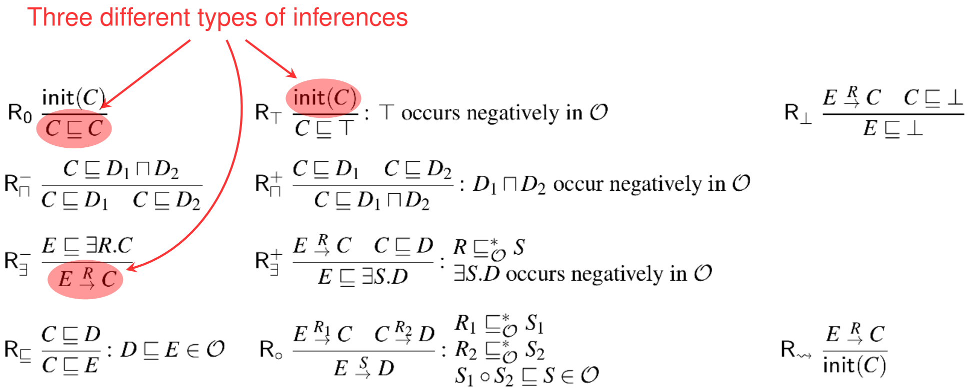
$$\mathsf{R}_\sqcap^-\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad \mathsf{R}_\sqcap^+\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occur negatively in } \mathcal{O}$$

$$\mathsf{R}_\exists^-\ \frac{E \sqsubseteq \exists R.C}{E \overset{R}{\rightarrow} C} \qquad \mathsf{R}_\exists^+\ \frac{E \overset{R}{\rightarrow} C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D} : \begin{array}{l} R \sqsubseteq^*_{\mathcal{O}} S \\ \exists S.D \text{ occurs negatively in } \mathcal{O} \end{array}$$

$$\mathsf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathsf{R}_\circ\ \frac{E \overset{R_1}{\rightarrow} C \quad C \overset{R_2}{\rightarrow} D}{E \overset{S}{\rightarrow} D} : \begin{array}{l} R_1 \sqsubseteq^*_{\mathcal{O}} S_1 \\ R_2 \sqsubseteq^*_{\mathcal{O}} S_2 \\ S_1 \circ S_2 \sqsubseteq S \in \mathcal{O} \end{array} \qquad \mathsf{R}_\rightsquigarrow\ \frac{E \overset{R}{\rightarrow} C}{\mathsf{init}(C)}$$

Fig. 3 Optimized inference rules for classification of $\mathcal{EL}_\bot^+$ ontologies

Encoding a calculus in rules



Three different types of inferences

$$\mathsf{R}_0\ \frac{\mathsf{init}(C)}{C \sqsubseteq C} \qquad \mathsf{R}_\top\ \frac{\mathsf{init}(C)}{C \sqsubseteq \top} : \top \text{ occurs negatively in } \mathcal{O} \qquad \mathsf{R}_\bot\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq \bot}{E \sqsubseteq \bot}$$

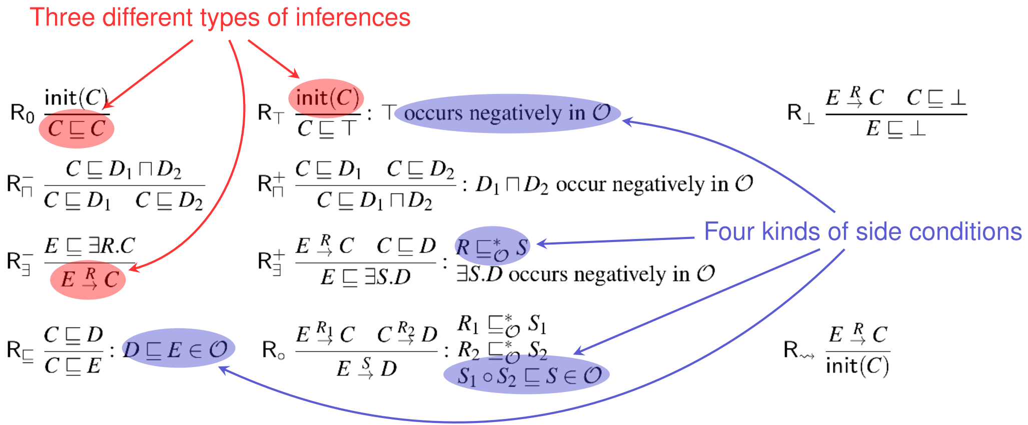
$$\mathsf{R}_\sqcap^-\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad \mathsf{R}_\sqcap^+\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occur negatively in } \mathcal{O}$$

$$\mathsf{R}_\exists^-\ \frac{E \sqsubseteq \exists R.C}{E \xrightarrow{R} C} \qquad \mathsf{R}_\exists^+\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D} : \begin{array}{l} R \sqsubseteq^*_{\mathcal{O}} S \\ \exists S.D \text{ occurs negatively in } \mathcal{O} \end{array}$$

$$\mathsf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathsf{R}_\circ\ \frac{E \xrightarrow{R_1} C \quad C \xrightarrow{R_2} D}{E \xrightarrow{S} D} : \begin{array}{l} R_1 \sqsubseteq^*_{\mathcal{O}} S_1 \\ R_2 \sqsubseteq^*_{\mathcal{O}} S_2 \\ S_1 \circ S_2 \sqsubseteq S \in \mathcal{O} \end{array} \qquad \mathsf{R}_\rightsquigarrow\ \frac{E \xrightarrow{R} C}{\mathsf{init}(C)}$$

Fig. 3 Optimized inference rules for classification of $\mathcal{EL}^+_\bot$ ontologies

Encoding a calculus in rules



$$\mathsf{R}_0\ \frac{\mathsf{init}(C)}{C \sqsubseteq C} \qquad \mathsf{R}_\top\ \frac{\mathsf{init}(C)}{C \sqsubseteq \top} : \top \text{ occurs negatively in } \mathcal{O} \qquad \mathsf{R}_\bot\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq \bot}{E \sqsubseteq \bot}$$

$$\mathsf{R}_\sqcap^-\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad \mathsf{R}_\sqcap^+\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occur negatively in } \mathcal{O}$$

$$\mathsf{R}_\exists^-\ \frac{E \sqsubseteq \exists R.C}{E \xrightarrow{R} C} \qquad \mathsf{R}_\exists^+\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D} : \begin{array}{l} R \sqsubseteq^*_{\mathcal{O}} S \\ \exists S.D \text{ occurs negatively in } \mathcal{O} \end{array}$$

$$\mathsf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathsf{R}_\circ\ \frac{E \xrightarrow{R_1} C \quad C \xrightarrow{R_2} D}{E \xrightarrow{S} D} : \begin{array}{l} R_1 \sqsubseteq^*_{\mathcal{O}} S_1 \\ R_2 \sqsubseteq^*_{\mathcal{O}} S_2 \\ S_1 \circ S_2 \sqsubseteq S \in \mathcal{O} \end{array} \qquad \mathsf{R}_\rightsquigarrow\ \frac{E \xrightarrow{R} C}{\mathsf{init}(C)}$$

Fig. 3 Optimized inference rules for classification of $\mathcal{EL}^{+}_{\bot}$ ontologies

Encoding expressions in predicates

We simply turn every expression in the calculus into a fact:

Expression in calculus	**Encoding in Datalog facts**
C occurs negatively in $\mathcal{O}$	`nf:isSubClass(`C`)`
$C \sqsubseteq D \in \mathcal{O}$	`nf:subClassOf(`C`,`D`)`
$R \sqsubseteq^*_{\mathcal{O}} S$	`nf:subProOf(`R`,`S`)`
$S_1 \circ S_2 \sqsubseteq S$	`nf:subPropChain(`S_1`,`S_2`,`S`)`
$C \sqsubseteq D$	`inf:subClassOf(`C`,`D`)`
$E \xrightarrow{R} C$	`inf:ex(`E`,`R`,`C`)`
init(C)	`inf:init(`C`)`

Encoding class expressions

We also need to encode the structure of class expressions

Encoding class expressions

We also need to encode the structure of class expressions

We use an obvious encoding where every sub-expression becomes a fact.

Example: The class $A \sqcap \exists R.(B \sqcap C)$ is encoded by facts

$$\texttt{nf:conj(}"A \sqcap \exists R.(B \sqcap C)"\texttt{,}A\texttt{,}"\exists R.(B \sqcap C)"\texttt{)}$$

$$\texttt{nf:exists(}"\exists R.(B \sqcap C)"\texttt{,}R\texttt{,}"B \sqcap C"\texttt{)}$$

$$\texttt{nf:conj(}"B \sqcap C"\texttt{,}B\texttt{,}C\texttt{)}$$

where every sub-expression is represented by a constant.

Expressions $\top$ and $\bot$ are encoded by their special OWL names `owl:Thing` and `owl:Nothing`.

Encoding expressions in predicates

Expression in calculus	**Encoding in Datalog facts**
$\top$	`owl:Thing`
$\bot$	`owl:Nothing`
$X = \exists R.C$	`nf:exists(`X`,`R`,`C`)`
$X = C \sqcap D$	`nf:conj(`X`,`C`,`D`)`
C occurs negatively in $\mathcal{O}$	`nf:isSubClass(`C`)`
$C \sqsubseteq D \in \mathcal{O}$	`nf:subClassOf(`C`,`D`)`
$R \sqsubseteq^*_{\mathcal{O}} S$	`nf:subProOf(`R`,`S`)`
$S_1 \circ S_2 \sqsubseteq S$	`nf:subPropChain(`S_1`,`S_2`,`S`)`
$C \sqsubseteq D$	`inf:subClassOf(`C`,`D`)`
$E \xrightarrow{R} C$	`inf:ex(`E`,`R`,`C`)`
init(C)	`inf:init(`C`)`

Encoding calculus rules in Datalog

Now all rules from the paper can simply be transcoded

Example:

$$\mathsf{R}^{+}_{\sqcap}\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occur negatively in } \mathcal{O}$$

becomes

```
inf:subClassOf(?C,?D1andD2) :-
    inf:subClassOf(?C,?D1), inf:subClassOf(?C,?D2),
    nf:conj(?D1andD2,?D1,?D2), nf:isSubClass(?D1andD2) .
```

Bringing it all together

Steps to produce the Datalog rules:

1. Read the paper carefully and understand the rule structure
2. Define predicates to encode the relevant expressions
3. Rewrite the rules in the new language

Steps to classify an ontology:

1. Encode the ontology using facts for the `nf:` predicates
2. Store the facts in an rls file, or in csv files
3. Evaluate this data with the calculus rules
4. Computed subclass relations are in predicate `inf:subClassOf`

Hands-On #4: Classifying Galen-EL

Let’s classify the Galen ontology (EL version)

(1) `@clear ALL .` (if still running)

(2) Register normalised Galen sources and load calculus:

```
@load "el/galen-sources.rls" .
@load "el/elk-calculus.rls" .
```

(3) `@reason .`

(4) Try some queries:

```
@query COUNT mainSubClassOf(?A,?B) .
@query mainSubClassOf(?A,galen:Virus) .
```

(5) Export classification to file:

```
@query mainSubClassOf(?A,?B) EXPORTCSV "galen-inf-subclass.csv" .
```

Performance tuning

Performance is ok for a first translation, but could be improved . . .

. . . but effective tuning requires knowledge of the reasoner!

Performance tuning

Performance is ok for a first translation, but could be improved ...
... but effective tuning requires knowledge of the reasoner!

Special aspects of VLog:

- Predicate tuples are indexed in their given order
  Fast: `p(?X,?Y,?Z), q(?X,?Y,?V)`
  Slow: `p(?Z,?Y,?X), q(?V,?X,?Y)`
- Body conjunctions are evaluated using binary joins
- Join order is determined by heuristics (esp. predicate size)
  Fast: short bodies; selective binary joins
  Slow: long bodies; possibly very un-selective joins

Running in VLog in debug-mode can yield insights on slow rule executions.

Performance tuning 1: Decompose rules

Some rules are hard to process:

```
inf:subClassOf(?E,?Y) :- inf:ex(?E,?R,?C), inf:subClassOf(?C,?D),
      nf:subProp(?R,?S), nf:exists(?Y,?S,?D), nf:isSubClass(?Y) .
```

Performance tuning 1: Decompose rules

Some rules are hard to process:

```
inf:subClassOf(?E,?Y) :- inf:ex(?E,?R,?C), inf:subClassOf(?C,?D),
        nf:subProp(?R,?S), nf:exists(?Y,?S,?D), nf:isSubClass(?Y) .
```

Likely bad join order (starting from small predicates):

$$(\texttt{nf:exists(?Y,?S,?D)} \bowtie \texttt{nf:subProp(?R,?S)}) \bowtie \texttt{inf:ex(?E,?R,?C)}$$

But most ontologies have very few properties (`?R`, `?S`), each used in a large part of the existential restrictions ⇝ essentially a product `nf:exists(?Y,?S,?D)` × `inf:ex(?E,?R,?C)`

Performance tuning 1: Decompose rules

Some rules are hard to process:

```
inf:subClassOf(?E,?Y) :- inf:ex(?E,?R,?C), inf:subClassOf(?C,?D),
      nf:subProp(?R,?S), nf:exists(?Y,?S,?D), nf:isSubClass(?Y) .
```

Likely bad join order (starting from small predicates):

$$(\texttt{nf:exists(?Y,?S,?D)} \bowtie \texttt{nf:subProp(?R,?S)}) \bowtie \texttt{inf:ex(?E,?R,?C)}$$

But most ontologies have very few properties (`?R`, `?S`), each used in a large part of the existential restrictions ⇝ essentially a product `nf:exists(?Y,?S,?D)` × `inf:ex(?E,?R,?C)`

Solution: Replace problematic rule by several rules:

```
     subExt(?D,?R,?Y) :- nf:subProp(?R,?S), nf:exists(?Y,?S,?D),
                         nf:isSubClass(?Y) .
        aux(?C,?R,?Y) :- inf:subClassOf(?C,?D), subExt(?D,?R,?Y) .
inf:subClassOf(?E,?Y) :- inf:ex(?E,?R,?C), aux(?C,?R,?Y) .
```

Performance tuning 2: Argument order

Argument order in derived predicates can be changed:

```
inf:subClassOf(?E,?Y) :- inf:ex(?E,?R,?C), aux(?C,?R,?Y) .
```

Performance tuning 2: Argument order

Argument order in derived predicates can be changed:

```
inf:subClassOf(?E,?Y) :- inf:ex(?E,?R,?C), aux(?C,?R,?Y) .
```

For this rule, it would work better if we flipped the order of `inf:ex`:

```
inf:subClassOf(?E,?Y) :- inf:xe(?C,?R,?E), aux(?C,?R,?Y) .
```

Of course, this must be done across all rules!

Performance tuning 2: Argument order

Argument order in derived predicates can be changed:

```
inf:subClassOf(?E,?Y) :- inf:ex(?E,?R,?C), aux(?C,?R,?Y) .
```

For this rule, it would work better if we flipped the order of `inf:ex`:

```
inf:subClassOf(?E,?Y) :- inf:xe(?C,?R,?E), aux(?C,?R,?Y) .
```

Of course, this must be done across all rules!

An optimised version of the calculus is in file `el/elk-caclulus-optimised.rls`.
Try it with Galen.

General guideline: There is no simple rule for how to improve performance, since many optimisations interact. Try what works best.
(The fastest results come from making typos: be sure to check correctness, too!)

Normalisation

The calculus requires us to pre-compute facts for the ontology encoding

- Standard libraries like the OWL API for Java can help
- But it still requires another software tool

Normalisation

The calculus requires us to pre-compute facts for the ontology encoding

- Standard libraries like the OWL API for Java can help
- But it still requires another software tool

Can’t we do this in rules, too?

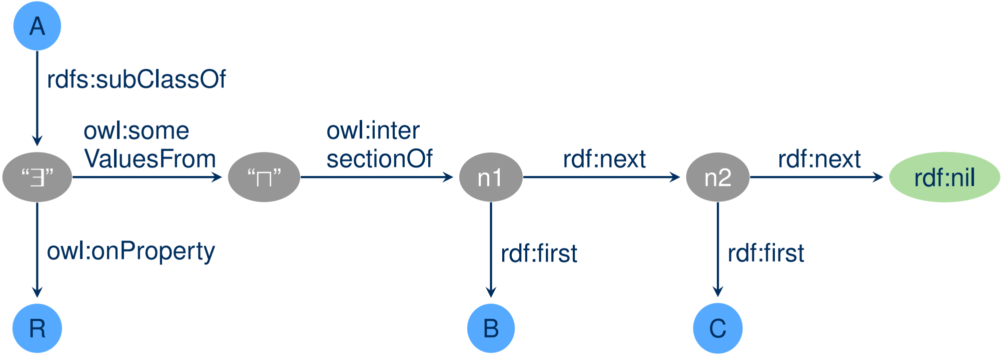
Rationale:

- OWL (DL) ontologies are typically stored in an RDF encoding
- Rulewerk and VLog can read RDF data natively
- Rules can perform structural transformations

$\mathcal{EL}$ in RDF

The RDF format describes labelled graphs, and DL axioms are encoded in graphs as well.

The following graph encodes $A \sqsubseteq \exists R.(B \sqcap C)$**:**



Extracting $\mathcal{EL}$ from RDF

Observation: OWL/RDF contains enough auxiliary nodes to use to represent subexpressions!

Extracting $\mathcal{EL}$ from RDF

Observation: OWL/RDF contains enough auxiliary nodes to use to represent subexpressions!

Making suitable rules is not hard:

- Extracting $C \sqsubseteq D$:

```
nf:subClassOf(?C,?D) :- TRIPLE(?C, rdfs:subClassOf, ?D) .
```

- Extracting $\exists R.X$:

```
nf:exists(?X,?R,?C) :- TRIPLE(?X, owl:someValuesFrom, ?C),
                       TRIPLE(?X, owl:onProperty, ?R) .
```

- Extracting binary $B \sqcap C$:

```
ex:conj(?X,?B,?C) :-
    TRIPLE(?X, owl:intersectionOf, ?L1),
    TRIPLE(?L1,rdf:next,?L2), TRIPLE(?L2,rdf:next,rdf:nil),
    TRIPLE(?L1,rdf:first,?B), TRIPLE(?L2,rdf:first,?C) .
```

The general case requires some more rules, since OWL encodes n-ary conjunctions as linked lists.

Reusing sub-expressions

Problem: The same class expression can occur thousands of times in one ontology
$\leadsto$ duplicated structures, which will all be inferred to be equivalent!

Reusing sub-expressions

Problem: The same class expression can occur thousands of times in one ontology
$\leadsto$ duplicated structures, which will all be inferred to be equivalent!

Solution: Replace auxiliary nodes by new elements, unique for each expression

Reusing sub-expressions

Problem: The same class expression can occur thousands of times in one ontology
⇝ duplicated structures, which will all be inferred to be equivalent!

Solution: Replace auxiliary nodes by new elements, unique for each expression

Approach:

- Mark the “main classes” that are not used in auxiliary positions (using negation)
- Use auxiliary predicates for syntactic extraction, e.g.:

```
synEx(?X,?R,?C) :- TRIPLE(?X, owl:someValuesFrom, ?C),
                   TRIPLE(?X, owl:onProperty, ?R) .
```

- Create and define representatives for every expression, recursively:

```
          repOf(?X,?X) :- nf:isMainClass(?X) .
   synExRep(?X,?R,?Rep) :- synEx(?X,?R,?Y), repOf(?Y,?Rep) .
nf:exists(!New,?R,?Rep) :- synExRep(?X,?R,?Rep) .
          repOf(?X,?N) :- synExRep(?X,?R,?Rep), nf:exists(?N,?R,?Rep) .
```

Hands-On #5: Normalising Galen

Rules for OWL $\mathcal{EL}$ normalisation are given in `el/elk-normalisation.rls`

Steps to normalise Galen EL from OWL/RDF

1. `@clear ALL .` (if still running)
2. Load Galen from RDF:
   ```
   @load RDF "el/galen-el.rdf" .
   ```
3. Load the normalisation rules:
   ```
   @load "el/elk-normalisation.rls" .
   ```
4. `@reason .`
5. Check result, e.g.,
   ```
   @query nf:exists(?X,?R,?C) LIMIT 10 .
   ```
6. Export normalised facts to CSV, e.g.,
   ```
   @query nf:subClassOf(?C,?D) EXPORTCSV "my-galen-subClassOf.csv" .
   ```

Putting it all together

We have just implemented a complete $\mathcal{EL}$ reasoner in 46 existential rules:
just load `elk-normalisation.rls` and `elk-calculus-optimised.rls` together with the triples of a OWL/RDF file!

Putting it all together

We have just implemented a complete $\mathcal{EL}$ reasoner in 46 existential rules:
just load `elk-normalisation.rls` and `elk-calculus-optimised.rls` together with the triples of a OWL/RDF file!

How about performance?

- Running normalisation and reasoning separately is faster than doing everything in one step (more rules – harder to optimise for VLog)
- Performance is below dedicated OWL EL reasoners, but practical:

[Laptop, Intel i7 2.70GHz, 4G Java heap]	Normalisation only	Reasoning only	All in one
GALEN EL (250K triples)	2.5sec	25sec	4min
SNOMED CT (2.9M triples)	30sec	2min	9min

But then again, this only took <50 lines of code!

Summary

What we learned

- Many rules-based reasoning calculi can be implemented in rules
- This is a multi-step process:
  - Develop suitable encoding
  - Translate and debug rules
  - Optimise performance
- Rules also help with related tasks (normalisation, reduction, result comparison, . . .)
- Rulewerk/VLog can be used for rapid prototyping of reasoning calculi

Up next: how to handle reasoning tasks beyond P

References

[1] David Carral, Irina Dragoste, Larry González, Ceriel J. H. Jacobs, Markus Krötzsch, Jacopo Urbani: **VLog: A Rule Engine for Knowledge Graphs.** ISWC (2) 2019: 19-35 Current main reference for Rulewerk (formerly: VLog4j)

[2] David Carral, Irina Dragoste, Markus Krötzsch: **Reasoner = Logical Calculus + Rule Engine.** KI - Künstliche Intelligenz, 2020. Further discussion of this use case (rules for reasoning)

[3] Yevgeny Kazakov, Markus Krötzsch, Frantisek Simancik: **The Incredible ELK – From Polynomial Procedures to Efficient Reasoning with $\mathcal{EL}$ Ontologies.** J. Autom. Reason. 53(1): 1-61 (2014) Source of the DL reasoning calculus used herein

[4] Markus Krötzsch: **Efficient Rule-Based Inferencing for OWL EL.** IJCAI 2011: 2668-2673 An earlier, less efficient Datalog calculus for $\mathcal{EL}$

[5] Jacopo Urbani, Ceriel J. H. Jacobs, Markus Krötzsch: **Column-Oriented Datalog Materialization for Large Knowledge Graphs.** AAAI 2016: 258-264 Original publication about VLog's design; explains the indexing method relevant for our optimisation here