

Oxford University
Computing Laboratory



Efficient Reasoning for OWL EL



Markus Krötzsch

Oxford University Computing Laboratory

Karlsruhe Institute of Technology (KIT)

Paper published at Jelia 2010 → [further details & download](#)

September 14, 2010



OWL EL

- Knowledge Representation on the Semantic Web: OWL 2
 - W3C standard for Web Ontology Language
 - OWL EL: one of the three lightweight “OWL profiles”
 - Related description logic: $SR\mathcal{OEL}(\Pi, \times)$
- **Goal 1: Specify PTime reasoning algorithm for OWL EL**
- **Goal 2: Quantify “difficulty” of efficiently implementing OWL EL features in reasoners**

A Simple Description Logic: $\mathcal{EL}\mathcal{O}$

- DLs are a family of knowledge representation languages
- Building blocks:
individuals I, **concepts A** (unary pred.), **roles R** (binary pred.)
- Concept expressions of $\mathcal{EL}\mathcal{O}$:

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

- “Axioms” (sentences) in $\mathcal{EL}\mathcal{O}$:

$C1 \sqsubseteq C2$ (concept inclusion)

$C(a)$ (concept assertion, $\{a\} \sqsubseteq C$)

$R(a,b)$ (role assertion, $\{a\} \sqsubseteq \exists R.\{b\}$)

Instance Retrieval using Datalog

- Example:

$C(a)$

$C \sqsubseteq D$

$C \sqcap D \sqsubseteq E$

- Write DL axioms as datalog facts:

SubClass(a, C)
Nom(a)

SubClass(C, D)

SubConj(C, D, E)

- Relevant deduction rules:

Nom(x) \rightarrow inst(x, x)

SubClass(y, z) \wedge inst(x, y) \rightarrow inst(x, z)

SubConj(y_1, y_2, z) \wedge inst(x, y_1) \wedge inst(x, y_2) \rightarrow inst(x, z)

An Instance Retrieval Calculus for $\mathcal{EL}\mathcal{O}$

Rules for a sound & complete instance retrieval calculus for $\mathcal{EL}\mathcal{O}$:

$$\mathbf{Nom}(x) \rightarrow \text{inst}(x,x)$$

$$\mathbf{SubClass}(y,z) \wedge \text{inst}(x,y) \rightarrow \text{inst}(x,z)$$

$$\mathbf{SubConj}(y_1,y_2,z) \wedge \text{inst}(x,y_1) \wedge \text{inst}(x,y_2) \rightarrow \text{inst}(x,z)$$

$$\mathbf{SupEx}(y,v,z,x') \rightarrow \text{inst}(x',z)$$

$$\mathbf{SupEx}(y,v,z,x') \wedge \mathbf{SubEx}(v,y',z') \wedge \text{inst}(x,y) \wedge \text{inst}(x',y') \rightarrow \text{inst}(x,z')$$

$$\text{inst}(x,y) \wedge \mathbf{Nom}(y) \wedge \text{inst}(x,z) \rightarrow \text{inst}(y,z)$$

$$\text{inst}(x,y) \wedge \mathbf{Nom}(y) \wedge \text{inst}(y,z) \rightarrow \text{inst}(x,z)$$

From Instance Retrieval to Classification

- 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

- 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!**
- Idea: Record assumption that leads to an inference:

inst(x,y) becomes inst(x,y,A)

A Classification Calculus for $\mathcal{EL}\mathcal{O}$

$$\begin{aligned} & \mathbf{Class}(q) \rightarrow \text{inst}(q,q,q) \\ & \mathbf{Nom}(x) \wedge \mathbf{Class}(q) \rightarrow \text{inst}(x,x,q) \\ & \mathbf{SubClass}(y,z) \wedge \text{inst}(x,y,q) \rightarrow \text{inst}(x,z,q) \\ & \mathbf{SubConj}(y_1,y_2,z) \wedge \text{inst}(x,y_1,q) \wedge \text{inst}(x,y_2,q) \rightarrow \text{inst}(x,z,q) \\ & \mathbf{SupEx}(y,v,z,x') \wedge \mathbf{Class}(q) \rightarrow \text{inst}(x',z,q) \\ & \mathbf{SupEx}(y,v,z,x') \wedge \mathbf{SubEx}(v,y',z') \wedge \text{inst}(x,y,q) \wedge \text{inst}(x',y',q) \rightarrow \text{inst}(x,z',q) \\ & \text{inst}(x,y,q) \wedge \mathbf{Nom}(y) \wedge \text{inst}(x,z,q) \rightarrow \text{inst}(y,z,q) \\ & \text{inst}(x,y,q) \wedge \mathbf{Nom}(y) \wedge \text{inst}(y,z,q) \rightarrow \text{inst}(x,z,q) \end{aligned}$$

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} :

Class(q) \rightarrow inst(q, q)

Nom(x) \rightarrow inst(x, x)

SubClass(y, z) \wedge inst(x, y) \rightarrow inst(x, z)

SubConj(y_1, y_2, z) \wedge inst(x, y_1) \wedge inst(x, y_2) \rightarrow inst(x, z)

SupEx(y, v, z, x') \rightarrow inst(x', z)

SupEx(y, v, z, x') \wedge **SubEx**(v, y', z') \wedge inst(x, y) \wedge inst(x', y') \rightarrow inst(x, z')

A Binary Classification Calculus for $\mathcal{EL}\mathcal{O}$?

- The problem with nominals:

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

→ $C_1 \sqsubseteq C_2$ follows **if A is non-empty.**

A Binary Classification Calculus for $\mathcal{EL}\mathcal{O}$?

- The problem with nominals:

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

→ $C_1 \sqsubseteq C_2$ follows **if A is non-empty.**

- We could cover this case with a new rule ...
... but would this be enough?

A Binary Classification Calculus for $\mathcal{EL}\mathcal{O}$?

- The problem with nominals:

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

→ $C_1 \sqsubseteq C_2$ follows **if A is non-empty.**

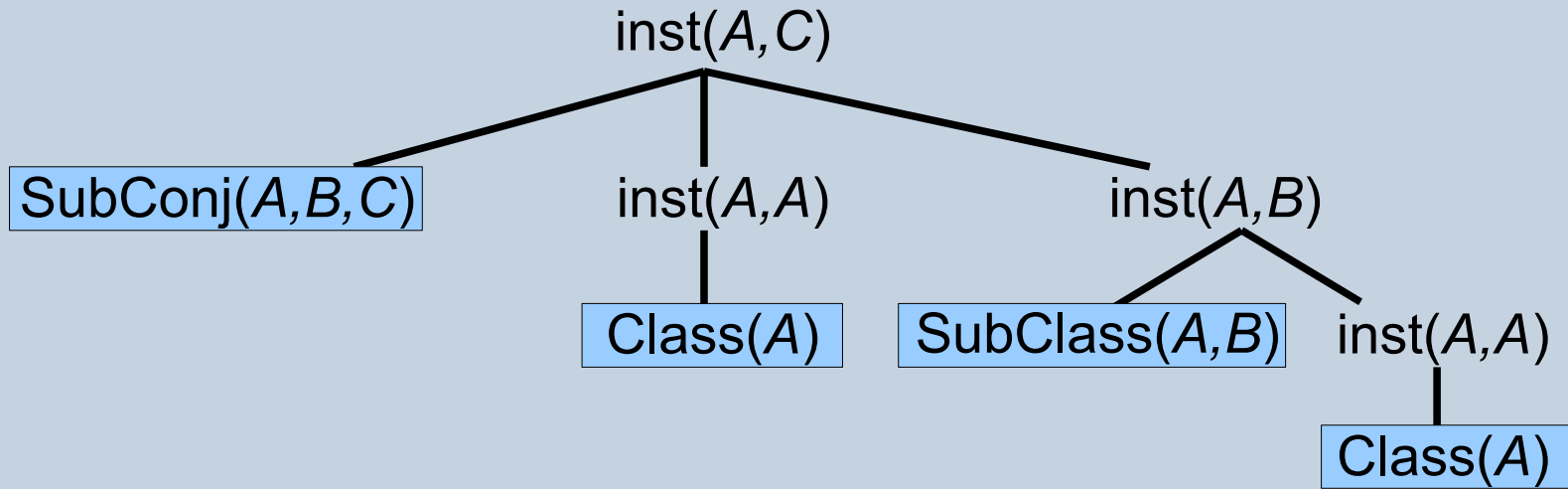
- We could cover this case with a new rule ...
... but would this be enough?

No!

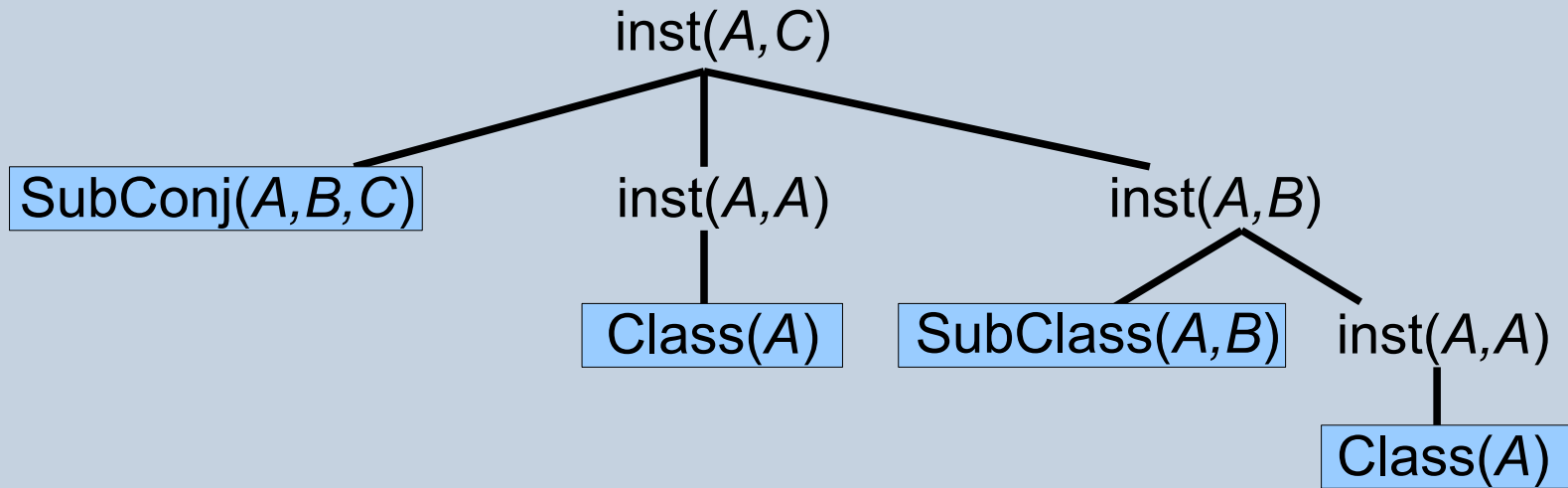
Proof Sketch

- **Claim:** If a classification calculus is sound and complete for \mathcal{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.
So the calculus must have a datalog proof tree for 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

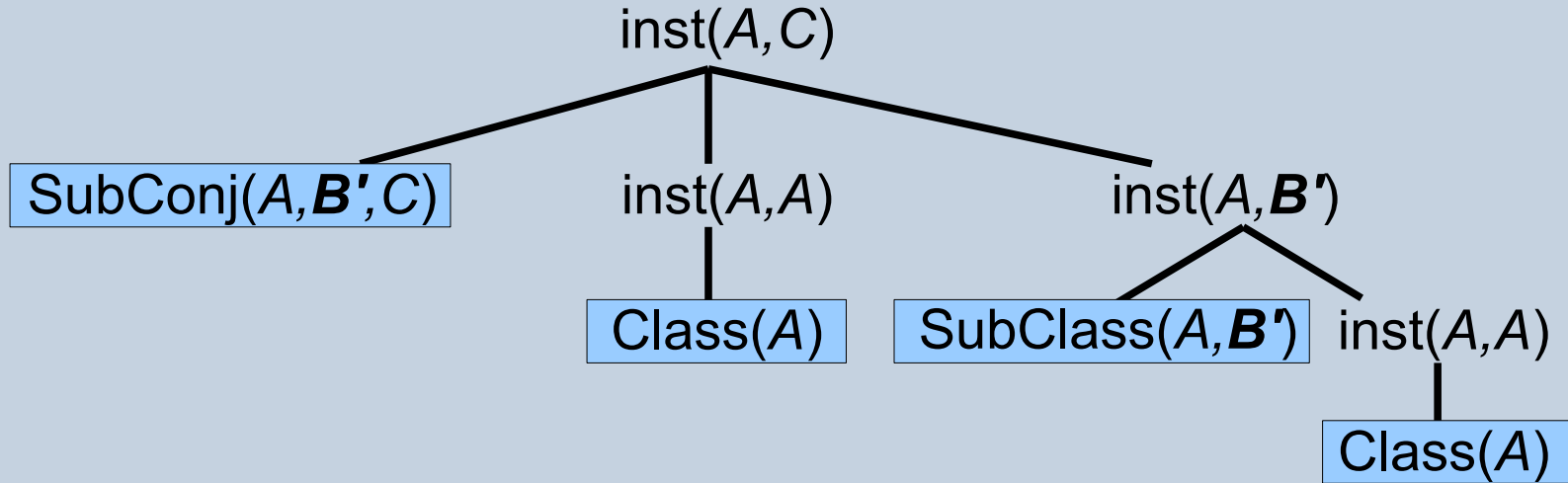


Transforming Proof Trees



- **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

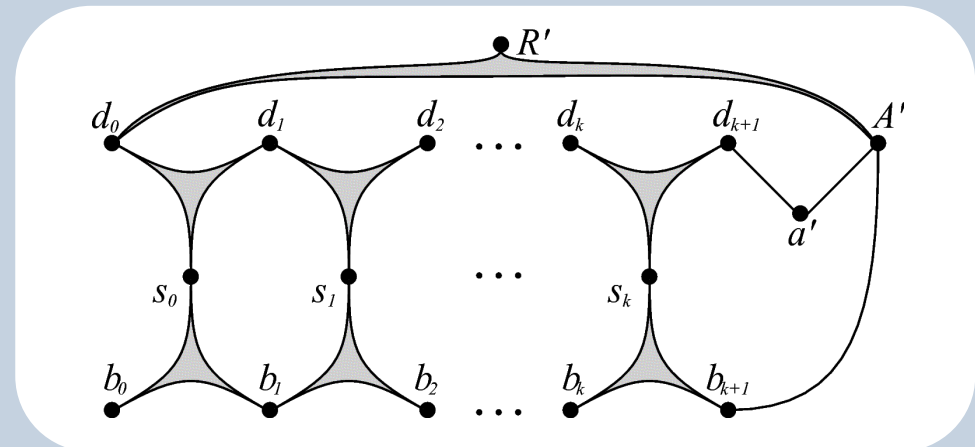
An Interesting Knowledge Base

- KB (for some $k > 0$):

$$\begin{array}{ll} \text{For } i=0, \dots, k: & D_i \sqsubseteq \exists S_i . D_{i+1} & \exists S_i . B_{i+1} \sqsubseteq B_i \\ & D_0 \sqsubseteq \exists R . A & A \sqsubseteq B_{k+1} \\ & D_{k+1} \sqsubseteq \{a\} & A \sqsubseteq \{a\} \end{array}$$

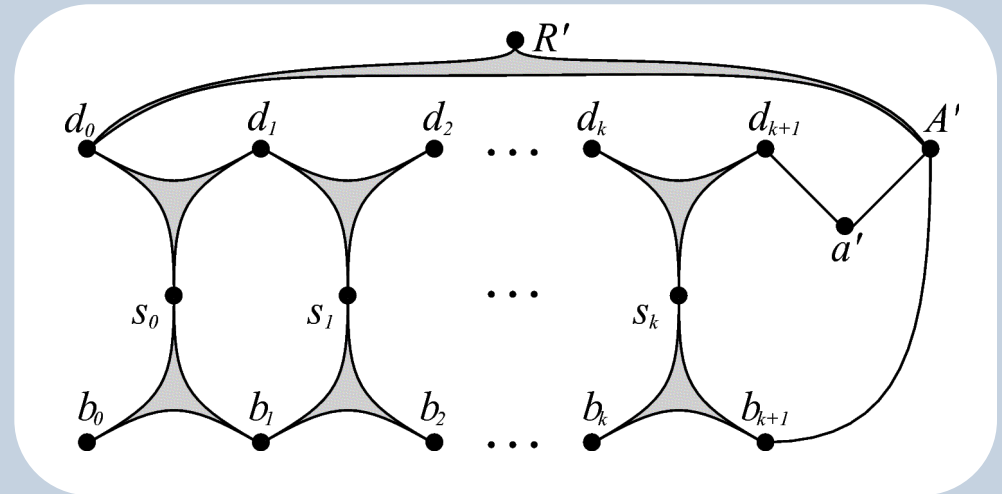
→ KB entails $D_0 \sqsubseteq B_0$

- Dependency graph:



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 →



Sub(proof)tree = Sub-knowledge-base = Sub(dependency)graph
 Symbols shared with rest of the tree = Symbols shared with axioms not in sub-KB = Nodes shared with rest of dependency graph

- 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.

Further Results and Conclusions

- Overview of results:

DL supports role chains?	–	yes	–	yes
DL supports nominals?	–	–	yes	yes
Minimal arity for instance retrieval	2	3	2	3
Minimal arity for classification	2	3	3	4

εLO OWL EL



- What this tells us:
 - Some DLs are more polynomial than others ;-)
 - Problematic features: role chains, nominals, universal (top) roles
 - Non-problematic features: Self, role conjunctions, range restrictions