Efficient Inferencing for OWL EL

Markus Krötzsch

Institute AIFB, Karlsruhe Institute of Technology, DE
mak@aifb.uni-karlsruhe.de

Abstract. We develop inferencing methods for $\mathcal{SROEL}(\sqcap, \times)$ – a DL that subsumes the main features of the W3C recommendation OWL EL –, and present a framework for studying materialisation calculi based on datalog. The latter is used to investigate the resource requirements for inferencing, and we can show that certain $\mathcal{SROEL}(\sqcap, \times)$ feature combinations must lead to increased space upper bounds in any materialisation calculus, suggesting that efficient implementations are easier to obtain for suitably chosen fragments of $\mathcal{SROEL}(\sqcap, \times)$.

1 Introduction

The recent OWL 2 W3C recommendation includes the lightweight ontology language OWL EL [9] which is semantically based on an extension of the $\mathcal{EL}^{++}$ description logic (DL). It is widely assumed that inferencing in OWL EL is possible in polynomial time, but it is not obvious how to extend existing reasoning procedures for $\mathcal{EL}^{++}$ accordingly [2]. In this paper, we set out to close this gap by developing suitable inferencing calculi for the DL $\mathcal{SROEL}(\sqcap, \times)$ which can be considered as an extension of the tractable DL $\mathcal{EL}^{++}$ with local reflexivity (Self), conjunctions of roles, and concept products. The latter two features generalise role disjointness, the universal (top) role, and admissible range restrictions as introduced in OWL EL. Concrete domains (datatypes) hardly interact with the additional features of $\mathcal{SROEL}(\sqcap, \times)$ and are not considered in this paper, though the according mechanisms used in [2] could be lifted to $\mathcal{SROEL}(\sqcap, \times)$.

Our second main contribution is to assess the *efficiency* of the proposed calculi. Inferencing for $\mathcal{EL}$-type DLs often suggests a materialisation-based (or consequence-driven) implementation, where all deductions are computed simultaneously in a bottom-up fashion. The number of inferable facts is an important measure of efficiency in this case, and we present a formalisation of materialisation calculi to relate it to the space complexity of datalog reasoning. Since upper space bounds for datalog are exponential in the *arity* of inferred predicates, our goal is to find materialisation calculi where these arities are low. We are able to show that there are limits to such optimisation: some inferencing tasks intrinsically require predicates of higher arities than others.

We present four inferencing calculi: a materialisation calculus for instance checking in $\mathcal{SROEL}(\sqcap, \times)$ in Section 3, and three calculi for classification in $\mathcal{SROEL}(\sqcap, \times)$ and two of its fragments in Section 4. Thereafter, in Section 5, we show that the arity of inferred predicates is minimal for each of the presented calculi. We provide extended sketches for some of the more interesting proofs to the extent that space permits. Detailed proofs for all results are found in the accompanying technical report [6].

Table 1. Syntax and semantics of $\mathcal{SROEL}(\sqcap,\times)$ axioms

Axiom	Syntax	Semantics for an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$R(a,b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
concept inclusion (GCI)	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role inclusion	$R \sqsubseteq T$	$R^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
generalised role inclusion	$R \circ S \sqsubseteq T$	$\{\langle x,z\rangle \mid \langle x,y\rangle \in R^{\mathcal{I}}, \langle y,z\rangle \in S^{\mathcal{I}} \text{ for some } y\} \subseteq T^{\mathcal{I}}$
role conjunction	$S_1 \sqcap S_2 \sqsubseteq T$	$S_1^{\mathcal{I}} \cap S_2^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
concept product	$C \times D \sqsubseteq T$	$C^{\mathcal{I}} \times D^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
	$R \sqsubseteq C \times D$	$T^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times D^{\mathcal{I}}$
$C, D \in \mathbf{C}$, $R, S_{(i)}, T \in \mathbf{N_R}$, $a, b \in \mathbf{N_I}$		

2 Preliminaries

This section summarises the basic notions from DL and datalog that are used in this paper. Readers who are not familiar with these topics may find extended introductory definitions in [6]. The main DL studied herein is $\mathcal{SROEL}(\sqcap,\times)$ which subsumes all semantic features of OWL EL that are not related to datatypes (concrete domains). $\mathcal{SROEL}(\sqcap,\times)$ is based on three disjoint finite sets of *individual names* $\mathbf{N_I}$, *concept names* $\mathbf{N_C}$, and *role names* $\mathbf{N_R}$. The set $\mathbf{C}$ of $\mathcal{SROEL}(\sqcap,\times)$ *concept expressions* then is given as $\mathbf{C} ::= \top \mid \bot \mid \mathbf{N_C} \mid \mathbf{C} \sqcap \mathbf{C} \mid \exists \mathbf{N_R}.\mathbf{C} \mid \exists \mathbf{N_R}.\mathsf{Self} \mid \{\mathbf{N_I}\}$. The set of $\mathcal{SROEL}(\sqcap,\times)$ *axioms* is defined as in Table 1. One may distinguish axioms of *ABox* (assertional axioms), *TBox* (terminological axioms: GCIs), and *RBox* (axioms related to roles).

Knowledge bases are sets of axioms that satisfy some additional properties. Consider a set KB of $\mathcal{SROEL}(\sqcap,\times)$ axioms. We inductively define the set of *non-simple roles* of KB to contain all roles T for which there is an axiom $R \circ S \sqsubseteq T \in$ KB, or an axiom $R \sqsubseteq T$ such that R is non-simple. A role that is not non-simple is called *simple*. Moreover, given a role name R, we define $\mathsf{ran}(R)$ to denote the set of concept expressions $D \in \mathbf{C}$ for which KB contains axioms $R \sqsubseteq S_1, \ldots, S_{n-1} \sqsubseteq S_n$ and $S_n \sqsubseteq C \times D$ for some $S_1, \ldots, S_n \in \mathbf{N_R}$ and $n \geq 0$. The set KB is a $\mathcal{SROEL}(\sqcap,\times)$ *knowledge base* if the following restrictions are satisfied:

- all roles S occurring in expressions $\exists S.\mathsf{Self} \in$ KB are simple,
- all roles S_1, S_2 occurring in axioms $S_1 \sqcap S_2 \sqsubseteq T \in$ KB are simple,
- for every axiom $R \circ S \sqsubseteq T \in$ KB we have $\mathsf{ran}(T) \subseteq \mathsf{ran}(S)$, and
- for every axiom $S_1 \sqcap S_2 \sqsubseteq T \in$ KB we have $\mathsf{ran}(T) \subseteq \mathsf{ran}(S_1) \cup \mathsf{ran}(S_2)$.

Note that we do not impose the structural restrictions of RBox regularity here [5] which also apply to OWL DL (and hence to OWL EL) ontologies, since these are not needed for efficient reasoning in $\mathcal{SROEL}(\sqcap,\times)$.

The semantics of $\mathcal{SROEL}(\sqcap,\times)$ is specified by defining DL interpretations $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ as usual. Here, we merely recall the semantics of axioms in Table 1; see [6] for a complete definition of $\mathcal{SROEL}(\sqcap,\times)$ semantics and entailment. Note that concept products on the left-hand side allow us to define the universal (top) role U with an axiom $\top \times \top \sqsubseteq U$. Since we can also define the empty (bottom) role N using $\exists N.\top \sqsubseteq \bot$, conjunctions of (simple) roles are a generalisation of disjointness of (simple) roles:

the axiom $R \sqcap S \sqsubseteq N$ declares S and R to be disjoint. In the absence of other role conjunctions, our requirements on concept products in $\mathcal{SROEL}(\sqcap, \times)$ knowledge bases agree with the known admissibility requirements for range restrictions in $\mathcal{EL}^{++}$ [3].

Our formalisation of inferencing calculi is based on the simple rule language *datalog* [1]. A *signature* of datalog is a tuple $\langle \mathbf{C}, \mathbf{P} \rangle$, where $\mathbf{C}$ is a finite set of *constants*, and $\mathbf{P}$ is a finite set of *predicates*, and each predicate $p \in \mathbf{P}$ has a fixed arity $\mathsf{ar}(p) \geq 0$. We assume $\mathbf{P}$ to be a disjoint union $\mathbf{P}_i \cup \mathbf{P}_e$ of *IDB predicates* $\mathbf{P}_i$ and *EDB predicates* $\mathbf{P}_e$.[1] A countably infinite set of *variables* is denoted by $\mathbf{V}$. Elements of $\mathbf{C} \cup \mathbf{V}$ are called *terms*.

A *datalog atom* over a signature $\langle \mathbf{C}, \mathbf{P} \rangle$ is an expression $p(t_1, \ldots, t_n)$ where $p \in \mathbf{P}$ with $\mathsf{ar}(p) = n$, and $t_i \in \mathbf{C} \cup \mathbf{V}$ for $i = 1, \ldots, n$. An IDB (EDB) atom is one that uses an IDB (EDB) predicate. A *datalog rule* is a formula of the form $B_1 \wedge \ldots \wedge B_l \rightarrow H$ where B_i and H are datalog atoms, and H is an IDB atom. The premise of a rule is also called its *body*, and the conclusion is called its *head*. A *datalog program* P is a set of datalog rules. A *fact* is a ground, i.e. variable-free, rule with an empty body.

A *ground substitution* σ for a signature $\langle \mathbf{C}, \mathbf{P} \rangle$ is a function $\sigma : \mathbf{V} \rightarrow \mathbf{C}$. Substitutions are extended to datalog atoms by setting $\sigma(p(t_1, \ldots, t_n)) := p(\sigma(t_1), \ldots, \sigma(t_n))$, and $\sigma(p(t_1, \ldots, t_n))$ is called a *ground instance* of $p(t_1, \ldots, t_n)$ in this case.

A *proof tree* for a datalog program P is a structure $\langle N, E, \lambda \rangle$ where N is a finite set of nodes, $E \subseteq N \times N$ is a set of edges of a directed tree, and λ is a labelling function that assigns a ground datalog atom to each node, where the following holds: for each node $n \in N$, there is a rule $B_1 \wedge \ldots \wedge B_l \rightarrow H \in P$ and a ground substitution σ such that $\lambda(n) = \sigma(H)$ and the set of child nodes $\{m \mid \langle n, m \rangle \in E\}$ is of the form $\{m_1, \ldots, m_l\}$ where $\lambda(m_i) = \sigma(B_i)$ for each $i = 1, \ldots, l$.

A ground atom H is a *consequence* of a datalog program P if there is a proof tree for P that has H as the label $\lambda(r)$ of its root node r.

Definition 1. *Given a datalog signature* $\langle \mathbf{C}, \mathbf{P} \rangle$*, a* renaming ρ *is a function* $\rho : \mathbf{C} \rightarrow \mathbf{C}$. *To extend* ρ *to ground datalog atoms we set* $\rho(p(t_1, \ldots, t_n)) := p(\rho(t_1), \ldots, \rho(t_n))$.

3 Instance Checking for $\mathcal{SROEL}(\sqcap, \times)$

We now introduce a calculus for solving the inference task of instance checking – deciding if $C(a)$ is entailed for any $C \in \mathbf{N_C}$, $a \in \mathbf{N_I}$ – for $\mathcal{SROEL}(\sqcap, \times)$. In Section 5 we show its optimality in the sense that no other materialisation calculus can be better in terms of certain characteristics. To prepare this study of calculi, it makes sense to seek a uniform presentation for deduction calculi that have been proposed for $\mathcal{EL}$-type DLs, e.g., in [2,4]. This motivates our use of datalog in this section.

Intuitively speaking, a materialisation calculus is a system of deduction rules for deriving logical consequences which – as opposed to a complete inferencing algorithm – does not specify a control flow or processing strategy for evaluating these rules. Deduction rules can be denoted in many forms, e.g. using textual if-then descriptions [2],

[1] This terminology originates from the field of deductive databases where one distinguishes *extensional* and *intensional data base*.

$C(a) \mapsto \{\texttt{subClass}(a, C)\}$	$R(a,b) \mapsto \{\texttt{subEx}(a, R, b, b)\}$	$a \in \mathbf{N_I} \mapsto \{\texttt{nom}(a)\}$
$\top \sqsubseteq C \mapsto \{\texttt{top}(C)\}$	$A \sqsubseteq \bot \mapsto \{\texttt{bot}(A)\}$	$A \in \mathbf{N_C} \mapsto \{\texttt{cls}(A)\}$
$\{a\} \sqsubseteq C \mapsto \{\texttt{subClass}(a, C)\}$	$A \sqsubseteq \{c\} \mapsto \{\texttt{subClass}(A, c)\}$	$R \in \mathbf{N_R} \mapsto \{\texttt{rol}(R)\}$
$A \sqsubseteq C \mapsto \{\texttt{subClass}(A, C)\}$	$A \sqcap B \sqsubseteq C \mapsto \{\texttt{subConj}(A, B, C)\}$	
$\exists R.\mathsf{Self} \sqsubseteq C \mapsto \{\texttt{subSelf}(R, C)\}$	$A \sqsubseteq \exists R.\mathsf{Self} \mapsto \{\texttt{supSelf}(A, R)\}$	
$\exists R.A \sqsubseteq C \mapsto \{\texttt{subEx}(R, A, C)\}$	$A \sqsubseteq \exists R.B \mapsto \{\texttt{supEx}(A, R, B, aux^{A \sqsubseteq \exists R.B})\}$	
$R \sqsubseteq T \mapsto \{\texttt{subRole}(R, T)\}$	$R \circ S \sqsubseteq T \mapsto \{\texttt{subRChain}(R, S, T)\}$	
$R \sqsubseteq C \times D \mapsto \{\texttt{supProd}(R, C, D)\}$	$A \times B \sqsubseteq R \mapsto \{\texttt{subProd}(A, B, R)\}$	
$R \sqcap S \sqsubseteq T \mapsto \{\texttt{subRConj}(R, S, T)\}$		
$A, B, C, D \in \mathbf{N_C}$, $R, S, T \in \mathbf{N_R}$, $a, b, c \in \mathbf{N_I}$		

Fig. 1. Input translation P_{inst}

in tabular form [9], or as sequent calculus style derivation rules [4]. Premises and conclusions of rules often consist of logical formulae, but may also contain auxiliary expressions that are relevant to the calculus.[2] A deduction rule can then be viewed as a schema for deriving new expressions from a finite set of given expressions. In particular, the applicability of rules is normally not affected by uniform renamings of signature symbols in premise and conclusion.

Deduction rules in this sense can be denoted as datalog rules where concrete logical sentences are represented as ground facts that use signature symbols in term positions. For example, we can represent $A \sqsubseteq B$ as $\texttt{subclassOf}(A, B)$, and introduce a rule $\texttt{subclassOf}(x, y) \wedge \texttt{subclassOf}(y, z) \rightarrow \texttt{subclassOf}(x, z)$. This unifies the presentation of diverse calculi, and allows us to exploit techniques from deductive databases. For connecting datalog to DL, we require an input translation from individual DL axioms to (sets of) datalog EDB facts. This translations is also defined for signature symbols, since symbols must generally be "loaded" into datalog to be able to derive conclusions about them, regardless of whether the symbols occurred in input axioms or not. A formalisation of these ideas is given later in Definition 2.

Calculi in the above sense generally suggest materialisation-based (or consequence-driven) reasoning: after translating a knowledge base to datalog facts, all consequences of these facts under the deduction rules can be computed in a bottom-up fashion, and all supported entailments can then be checked without further recursive computation. This contrasts with other reasoning principles such as the tableaux method where just a single entailment is checked in one run of the algorithm.

It is not hard to formulate the deduction algorithms presented for $\mathcal{EL}$-type logics in [2] and [4] using datalog rules. The calculus we present here, however, is derived from a datalog reduction introduced in [8] for a rule language based on $\mathcal{EL}^{++}$. This approach can be modified to cover $\mathcal{SROEL}(\sqcap, \times)$ and to use a fixed set of datalog rules to yield a materialisation calculus in our sense. For simplicity, the following calculus only considers $\mathcal{SROEL}(\sqcap, \times)$ axioms of the basic forms in Fig. 1. $\mathcal{SROEL}(\sqcap, \times)$ axioms can be translated to such normalised axioms in linear time so that all entailments of the input knowledge base are preserved [6].

[2] For instance, the calculus in [2] uses auxiliary statements $A \rightsquigarrow_R B$ for $A, B \in \mathbf{N_C}$.

(1)	$\texttt{nom}(x) \rightarrow \texttt{inst}(x,x)$
(2)	$\texttt{nom}(x) \wedge \texttt{triple}(x,v,x) \rightarrow \texttt{self}(x,v)$
(3)	$\texttt{top}(z) \wedge \texttt{inst}(x,z') \rightarrow \texttt{inst}(x,z)$
(4)	$\texttt{bot}(z) \wedge \texttt{inst}(u,z) \wedge \texttt{inst}(x,z') \wedge \texttt{cls}(y) \rightarrow \texttt{inst}(x,y)$
(5)	$\texttt{subClass}(y,z) \wedge \texttt{inst}(x,y) \rightarrow \texttt{inst}(x,z)$
(6)	$\texttt{subConj}(y_1,y_2,z) \wedge \texttt{inst}(x,y_1) \wedge \texttt{inst}(x,y_2) \rightarrow \texttt{inst}(x,z)$
(7)	$\texttt{subEx}(v,y,z) \wedge \texttt{triple}(x,v,x') \wedge \texttt{inst}(x',y) \rightarrow \texttt{inst}(x,z)$
(8)	$\texttt{subEx}(v,y,z) \wedge \texttt{self}(x,v) \wedge \texttt{inst}(x,y) \rightarrow \texttt{inst}(x,z)$
(9)	$\texttt{supEx}(y,v,z,x') \wedge \texttt{inst}(x,y) \rightarrow \texttt{triple}(x,v,x')$
(10)	$\texttt{supEx}(y,v,z,x') \wedge \texttt{inst}(x,y) \rightarrow \texttt{inst}(x',z)$
(11)	$\texttt{subSelf}(v,z) \wedge \texttt{self}(x,v) \rightarrow \texttt{inst}(x,z)$
(12)	$\texttt{supSelf}(y,v) \wedge \texttt{inst}(x,y) \rightarrow \texttt{self}(x,v)$
(13)	$\texttt{subRole}(v,w) \wedge \texttt{triple}(x,v,x') \rightarrow \texttt{triple}(x,w,x')$
(14)	$\texttt{subRole}(v,w) \wedge \texttt{self}(x,v) \rightarrow \texttt{self}(x,w)$
(15)	$\texttt{subRChain}(u,v,w) \wedge \texttt{triple}(x,u,x') \wedge \texttt{triple}(x',v,x'') \rightarrow \texttt{triple}(x,w,x'')$
(16)	$\texttt{subRChain}(u,v,w) \wedge \texttt{self}(x,u) \wedge \texttt{triple}(x,v,x') \rightarrow \texttt{triple}(x,w,x')$
(17)	$\texttt{subRChain}(u,v,w) \wedge \texttt{triple}(x,u,x') \wedge \texttt{self}(x',v) \rightarrow \texttt{triple}(x,w,x')$
(18)	$\texttt{subRChain}(u,v,w) \wedge \texttt{self}(x,u) \wedge \texttt{self}(x,v) \rightarrow \texttt{triple}(x,w,x)$
(19)	$\texttt{subRConj}(v_1,v_2,w) \wedge \texttt{triple}(x,v_1,x') \wedge \texttt{triple}(x,v_2,x') \rightarrow \texttt{triple}(x,w,x')$
(20)	$\texttt{subRConj}(v_1,v_2,w) \wedge \texttt{self}(x,v_1) \wedge \texttt{self}(x,v_2) \rightarrow \texttt{self}(x,w)$
(21)	$\texttt{subProd}(y_1,y_2,w) \wedge \texttt{inst}(x,y_1) \wedge \texttt{inst}(x',y_2) \rightarrow \texttt{triple}(x,w,x')$
(22)	$\texttt{subProd}(y_1,y_2,w) \wedge \texttt{inst}(x,y_1) \wedge \texttt{inst}(x,y_2) \rightarrow \texttt{self}(x,w)$
(23)	$\texttt{supProd}(v,z_1,z_2) \wedge \texttt{triple}(x,v,x') \rightarrow \texttt{inst}(x,z_1)$
(24)	$\texttt{supProd}(v,z_1,z_2) \wedge \texttt{self}(x,v) \rightarrow \texttt{inst}(x,z_1)$
(25)	$\texttt{supProd}(v,z_1,z_2) \wedge \texttt{triple}(x,v,x') \rightarrow \texttt{inst}(x',z_2)$
(26)	$\texttt{supProd}(v,z_1,z_2) \wedge \texttt{self}(x,v) \rightarrow \texttt{inst}(x,z_2)$
(27)	$\texttt{inst}(x,y) \wedge \texttt{nom}(y) \wedge \texttt{inst}(x,z) \rightarrow \texttt{inst}(y,z)$
(28)	$\texttt{inst}(x,y) \wedge \texttt{nom}(y) \wedge \texttt{inst}(y,z) \rightarrow \texttt{inst}(x,z)$
(29)	$\texttt{inst}(x,y) \wedge \texttt{nom}(y) \wedge \texttt{triple}(z,u,x) \rightarrow \texttt{triple}(z,u,y)$

Fig. 2. Deduction rules P_{inst}

Theorem 1. *Consider the materialisation calculus K_{inst} with input translation I_{inst} as in Fig. 1, and derivation rules P_{inst} as in Fig. 2. For a knowledge base* KB *such that $I_{inst}(\alpha)$ is defined for all $\alpha \in$ KB, set $P(\text{KB}) := P_{inst} \cup \bigcup_{\alpha \in \text{KB}} I_{inst}(\alpha) \cup \bigcup_{s \in \mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}} I_{inst}(s)$.*

For all $C \in \mathbf{N_C}$, and $a \in \mathbf{N_I}$, KB *entails $C(a)$ if and only if $P(\text{KB})$ entails $\texttt{inst}(a,C)$, whenever $P(\text{KB})$ is defined. Thus K_{inst} provides a materialisation calculus for instance checking for $\mathcal{SROEL}(\sqcap,\times)$ knowledge bases within which all axioms are normalised.*

The IDB predicates `inst`, `triple`, and `self` in P_{inst} correspond to ABox axioms for atomic concepts, roles, and concepts $\exists R.\mathsf{Self}$, respectively. Rule (1) serves as an initialisation rule that accounts for the first `inst` facts to be derived. Rule (2) specifies the (only) case where reflexive `triple` facts lead to `self` facts. The rules (3) to (26) capture expected derivations for each of the axiom types as encoded by the EDB predicates. Rule (4) checks for global inconsistencies, and would typically not be materialised in implementations since its effect can directly be taken into account during entailment checking. Rules (9) and (10) make use of auxiliary constants $aux^{A \sqsubseteq \exists R.B}$ for handling existentials. Roughly speaking, each such constant represents the class of all

role successors generated by the axiom from which it originates; see [6] for details. The remaining rules (27) to (29) encode equality reasoning that is relevant in the presence of nominals where statements $\texttt{inst}(a, b)$ with $a, b \in \mathbf{N_I}$ encode equality of a and b.

Axiom normalisation and the computation of I_{inst} can be accomplished in linear time, and the time for reasoning in datalog is polynomial w.r.t. the size of the collection of ground facts. Together with the known P-hardness of $\mathcal{EL}^{++}$ [2], we obtain the following result, of which no formal proof seems to have been published so far:

Corollary 1. *Instance checking in $\mathcal{SROEL}(\sqcap, \times)$ and in OWL EL without datatype properties is* P *complete w.r.t. the size of the knowledge base.*

This result can be extended to OWL EL with datatype properties along the lines of datatype reasoning in $\mathcal{EL}^{++}$ [2], but this is not implied by the above theorem. The proof of Theorem 1 is found in [6]. Completeness is obtained by transforming models of datalog programs to corresponding models of DL knowledge bases, part of which is to show that equality reasoning really suffices to establish a congruence between elements of the domain. Soundness is shown by interpreting the meaning of datalog atoms in terms of DL, and showing inductively that each rule application preserves soundness of this interpretation. This is most interesting for rules (19) and (25) where the result hinges upon the restrictions on role conjunction and concept products in $\mathcal{SROEL}(\sqcap, \times)$.

4 Classification of $\mathcal{SROEL}(\sqcap, \times)$ Knowledge Bases

The materialisation calculus K_{inst} of Theorem 1 solves the instance checking problem for $\mathcal{SROEL}(\sqcap, \times)$. A calculus for checking satisfiability is easily derived since a $\mathcal{SROEL}(\sqcap, \times)$ knowledge base is inconsistent if and only if K_{inst} infers a fact $\texttt{inst}(x, z)$ where $\texttt{bot}(z)$ holds. In this section, we ask how to obtain calculi for *classification* – the computation of all subsumptions of atomic classes implied by a knowledge base.

Class subsumption, too, can be reduced to instance retrieval: to check $A \sqsubseteq B$, one introduces a new individual c and adds an assertion $A(c)$; then the subsumption holds if the modified knowledge base entails $B(c)$. This reduction requires the knowledge base to be modified, leading to new entailments, possibly even to global inconsistency. Thus K_{inst} cannot directly be used for classification, since it is not feasible to introduce test individuals c for all (atomic) classes at load time so as to materialise all subsumptions in parallel. Rather, one would have to use a separate run of K_{inst} for each subclass A to compute all entailments of the form $A \sqsubseteq B$.

This approach allows us to derive a sound and complete materialisation calculus for materialisation in $\mathcal{SROEL}(\sqcap, \times)$ by "internalising" the runs of K_{inst} by extending all IDB predicates with an additional parameter to encode the test assumption under which this fact can be inferred. Our assumptions have the form $A(c)$, but the name of c is not essential. So we re-use the datalog constant A as the test instance of class A, such that the additional parameter of IDB atoms can simply be a concept name A. The proof of the following theorem is immediate from this discussion.

Theorem 2. *Consider the materialisation calculus K_{sc} with input translation I_{sc} defined like I_{inst} in Fig. 1 and datalog program P_{sc} containing the following rules:*

– *for each rule* $r \in P_{inst}$ *(Fig. 2), a rule* r' *obtained from* r *by adding a new body atom* $\texttt{cls}(q)$*, and replacing each IDB atom* $\texttt{inst}(x, y)$ *(*$\texttt{triple}(x, y, z)$*,* $\texttt{self}(x, y)$*) by an atom* $\texttt{inst\_sc}(x, y, q)$ *(*$\texttt{triple\_sc}(x, y, z, q)$*,* $\texttt{self\_sc}(x, y, q)$*), where* q *is a variable not occurring in* r*,*
– *the additional rule* $\texttt{cls}(q) \rightarrow \texttt{inst\_sc}(q, q, q)$*.*

For a knowledge base KB *such that* $I_{sc}(\alpha)$ *is defined for all* $\alpha \in$ KB*, set* $P(\text{KB}) := P_{sc} \cup \bigcup_{\alpha \in \text{KB}} I_{sc}(\alpha) \cup \bigcup_{s \in \mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}} I_{sc}(s)$*. Then for all* $A, B \in \mathbf{N_C}$*,* KB *entails* $A \sqsubseteq B$ *if and only if* $P(\text{KB})$ *entails* $\texttt{inst\_sc}(A, B, A)$*, whenever* $P(\text{KB})$ *is defined. Thus* K_{sc} *provides a materialisation calculus for subsumption checking for* $\mathcal{SROEL}(\sqcap, \times)$ *knowledge bases within which all axioms are normalised.*

It must be noted that K_{sc} is not very efficient since deductions that are globally true are inferred under each local assumption q independently. This means that the number of globally derived facts can multiply by the number of class names in the signature, e.g. by more than 300,000 for the popular SNOMED CT ontology. Our formalisation of materialisation calculi provides a direct measure of this increase: the maximal arity of IDB predicates in K_{sc} is four while it had been three in K_{inst}, leading to potentially higher space requirements for materialised derivations. Implementations may of course achieve lower space bounds by using suitable optimisations; yet standard implementation techniques for datalog, such as semi-naive materialisation, are sensitive to the number of parameters in IDB predicates. In developing the database-driven reasoner *Orel* [7], we also experienced major *runtime* penalties associated with higher arities due to the larger numbers of inferences that must be considered in each derivation step.

The arity of IDB predicates thus is an important measure for the efficiency of a materialisation calculus, and we will denote this parameter as the *arity of a calculus* and speak of binary/ternary/*n*-ary materialisation calculi. The search for more efficient materialisation calculi can now be formalised as the task of finding a ternary or binary calculus that is sound and complete for $\mathcal{SROEL}(\sqcap, \times)$ classification. Unfortunately, as shown in Section 5, such a calculus cannot exist. To illustrate that this is not obvious, we now present a classification calculus of lower arity for a fragment of $\mathcal{SROEL}(\sqcap, \times)$.

We now develop a ternary materialisation calculus that supports role chains but no $\top$, $\bot$, nominal classes, and concept products on the left-hand side of axioms. The input translation can remain as in Fig. 1 but without the cases that involve the excluded features. The EDB predicates `top`, `bot`, and `subProd` are no longer used.

A set of rules is developed by restricting the rules of K_{sc} of Theorem 2. We use the numbers as in Fig. 2 for referring to the rules obtained from K_{inst}. Rules (3), (4), (21), and (22) are no longer needed due to the restriction of EDB predicates. Without nominal classes, we find that all derivations $\texttt{inst\_sc}(x, y, q)$ are such that y is a DL class name, or y is a DL individual name and $x = y$. This is not hard to verify inductively by considering each rule, and the symbols used in relevant EDB facts. This shows that rules (27), (28), and (29) are obsolete as well. As shown in [6], the essential feature of the remaining rule set is that the additional parameter q that has been introduced for K_{sc} above is no longer required for obtaining a sound and complete materialisation calculus.

Theorem 3. *Consider the materialisation calculus K_{scc} with I_{scc} defined like I_{inst} in Fig. 1 but undefined for all axioms that use nominal classes, $\top$, $\bot$, or concept products on the left-hand side, and the program P_{scc} consisting of the rules* (1), (2), (5)–(20), *and* (23)–(26) *of Fig. 2 together with a new rule* $\texttt{cls}(z) \rightarrow \texttt{inst}(z, z)$.

For a knowledge base KB *such that $I_{scc}(\alpha)$ is defined for all $\alpha \in$ KB, set $P(\text{KB}) \coloneqq P_{scc} \cup \bigcup_{\alpha \in \text{KB}} I_{scc}(\alpha) \cup \bigcup_{s \in \mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}} I_{scc}(s)$. Then for all $A, B \in \mathbf{N_C}$,* KB *entails $A \sqsubseteq B$ if and only if P*(KB) *entails* $\texttt{inst}(A, B)$, *whenever P*(KB) *is defined. Thus K_{scc} provides a materialisation calculus for subsumption checking for $\mathcal{SROEL}(\sqcap, \times)$ knowledge bases that contain only $\sqcap$ (for concepts and roles), $\exists$, Self, $\circ$, and concept products on the right-hand side.*

In terms of OWL 2, the DL of the previous theorem covers all OWL EL ontologies without datatype properties and the constructs `owl:Thing`, `owl:topObjectProperty`, `owl:Nothing`, `owl:bottomObjectProperty`, `objectHasValue` and `objectOneOf`.

It is not hard to further simplify K_{scc} for the case that no role chains occur in the knowledge base, leading to a binary classification calculus for normalised $\mathcal{SROEL}(\sqcap, \times)$ knowledge bases that contain only $\sqcap$ (for concepts and roles), $\exists$, Self, and concept products on the right-hand side. For reasons of space, the calculus has been removed from the final version of this paper; it can still be found in [6]. A similar approach was used to optimise a classification calculus for $\mathcal{ELH}$ presented in [4].

5 Minimal Arities of Materialisation Calculi

The previously discussed materialisation calculi for $\mathcal{SROEL}(\sqcap, \times)$ featured different arities: while some reasoning tasks could be solved by binary and ternary calculi, our classification calculus for $\mathcal{SROEL}(\sqcap, \times)$ is 4-ary. We have argued above that lower arities are important for efficient processing, so it is desirable to develop materialisation calculi of minimal arity. In this section, we establish lower bounds on the arity of materialisation calculi for various reasoning problems. This requires a concrete understanding of what a materialisation calculus is. Generalising the properties of the calculi discussed above, we obtain the following formalisation of this notion.

Definition 2. *A* materialisation calculus *K is a tuple $K = \langle I, P, O \rangle$ where I and O are partial functions, and P is a set of datalog rules, such that*

1. *given an axiom or signature symbol α, $I(\alpha)$ is either undefined or a set of datalog facts over EDB predicates,*
2. *given an axiom α, $O(\alpha)$ is either undefined or a datalog fact over an IDB predicate,*
3. *the set of EDB and IDB predicates used by I, P, and O is fixed and finite,*
4. *P contains no constant symbols,*
5. *all constant symbols used in $I(\alpha)$ or $O(\alpha)$ for some axiom (or signature symbol) α are either signature symbols that appear in (or are equal to) α, or constants of the form aux_i^α with $i \geq 0$, where all constant names aux_i^α are mutually distinct and unequal to any DL signature symbol,*
6. *I and O do not depend on concrete signature symbols, i.e. for a renaming ρ of signature symbols that maps individual/concept/role names to individual/concept/role names, we find $I(\rho(\alpha)) = \rho(I(\alpha))$ and $O(\rho(\alpha)) = \rho(O(\alpha))$ if $\rho(aux_i^\alpha) = aux_i^{\rho(\alpha)}$.*

We extend I to knowledge bases KB *by setting* $I(\mathrm{KB}) := \bigcup_{\beta\in\mathrm{KB}} I(\beta)$ *if* $I(\beta)$ *is defined for all* $\beta \in$ KB *and undefined otherwise. We extend I to sets of signature symbols S by setting* $I(S) := \bigcup_{s\in S, I(s)\ defined} I(s)$*. K induces an* entailment relation $\vdash_K$ *between knowledge bases* KB *and axioms α over a signature* $\langle \mathbf{N_I}, \mathbf{N_C}, \mathbf{N_R}\rangle$*, defined by setting* $\mathrm{KB} \vdash_K \alpha$ *whenever* $I(\mathrm{KB})$ *and* $O(\alpha)$ *are defined and* $I(\mathrm{KB}) \cup I(\mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}) \cup P \models O(\alpha)$*.*

We say that K is sound (complete) *if* $\mathrm{KB} \vdash_K \alpha$ *implies (is implied by)* $\mathrm{KB} \models \alpha$ *for all knowledge bases* KB *and axioms α for which* $I(\mathrm{KB})$ *and* $O(\alpha)$ *are defined.*

Note that this definition explicitly allows the datalog transformation I to introduce arbitrarily many auxiliary constants aux_i^α. This can be utilised, e.g., to perform a normalisation that introduces auxiliary concept names as part of the input translation, or to introduce new constants for handling existentials as in the above calculi. Yet, the input translation is limited in its expressivity, since it depends only on individual axioms and signature symbols. In particular, this precludes complex datalog translations as in [10,11]. Note that we do not make any assumptions on the computability or complexity of I and O, but both functions are typically very simple.

Now our general proof strategy is as follows. For a contradiction, we suppose that there is a materialisation calculus of lower arity that solves a given reasoning problem. We then consider a particular instance of that problem, given by a knowledge base KB from which a relevant consequence α must follow. Since the calculus is assumed to be complete, we obtain an according datalog derivation with a corresponding proof tree. This proof tree is then modified by renaming constants, leading to a variant of the proof tree that is still valid for the given materialisation calculus, but that is based on different (renamed) assumptions. The modified assumptions correspond to a modified knowledge base KB′, and by our construction we find that the materialisation calculus still computes the entailment of α on the input KB′. We then show that α is not entailed by KB′, so that the calculus is proven to be unsound. Since KB′ is based on the modified proof tree, some graph theoretic arguments are required to establish this last step.

A central notion of this proof strategy is the following modification of proof trees.

Definition 3. *Consider a materialisation calculus $K = \langle I, P, O\rangle$ and a knowledge base* KB *such that* $I(\mathrm{KB})$ *is defined, and a proof tree $T = \langle N, E, \lambda\rangle$ for* $I(\mathrm{KB}) \cup I(\mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}) \cup P$*. We say that a DL signature symbol σ* occurs *in a ground atom F if F contains σ as a constant, or if F contains some auxiliary constant aux_i^α such that σ occurs in α. The* interface *of a node $n \in N$ is the set of signature symbols that occur in $\lambda(n)$.*

The (labels of) T can be diversified *by the following recursive construction:*

- *replace all signature symbols s that do not occur in the interface of the root node by a fresh symbol s' that has not yet been used in T or in this construction,*
- *recursively diversify the subtrees below each of the direct child nodes of the root.*

We tacitly assume that the datalog signature contains all required new constant names. Note that the renaming may affect auxiliary constants by renaming symbols in the axioms that are part of their name. The diversification is thus obtained by replacing some signature symbols with fresh symbols. This replacement may not be uniform throughout the tree, and we use s^n to denote the symbol by which s is replaced in node n.

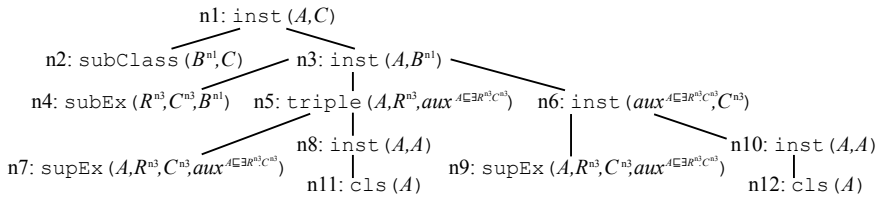

Fig. 3. Diversification of a K_{scc} proof for $\{A \sqsubseteq \exists R.C, \exists R.C \sqsubseteq B, B \sqsubseteq C\} \models A \sqsubseteq C$

Intuitively speaking, the above renaming removes any re-use of constant names throughout the proof tree that is not strictly necessary for applying the rules of P. What is "strictly necessary" is captured by the *interface* of each node: constants that are not in the interface of a rule application can be renamed uniformly in all descendants of the current node without affecting the correctness of the proof tree. This creates directly connects the arity of a calculus to the amount of renaming during diversification.

Figure 3 shows an example diversification based on the calculus K_{scc} of Theorem 3, where we use the notation from Definition 3 for denoting renamed symbols. Note how C is renamed to C^{n3} in some but not in all labels. Also note that no further renamings occur below the nodes $n5$ and $n6$ since all relevant symbols occur in their interface due to the auxiliary constant. As expected, the diversification is again a proof tree for a knowledge base that contains suitably renamed axioms:

Definition 4. *Consider a materialisation calculus K, knowledge base* KB*, and proof tree T as in Definition 3. Let λ' denote a diversified labelling for T.*

Let $m \in N$ be a leaf node with $\lambda(m) \in I(\alpha)$ for some $\alpha \in$ KB. By Definition 2, one can rename symbols in α to obtain an axiom α' such that $\lambda'(m) \in I(\alpha')$. Concretely, α' is obtained from α be replacing all symbols s in the interface of m by s^m, and by replacing all other symbols t by some fresh symbol t' not used anywhere yet. We select one such axiom α'_m for each such node m.

The diversification KB′ *of* KB *is the knowledge base* $\text{KB}' := \{\alpha'_n \mid n \in N, n \text{ a leaf}\}$. *The tree structure of T can be used to represent* KB′ *as a set of nested sets Γ_n for $n \in N$, recursively defined by setting $\Gamma_n := \{\alpha'_m \mid \langle n, m\rangle \in E, m \text{ a leaf}\} \cup \{\Gamma_m \mid \langle n, m\rangle \in E, m \text{ not a leaf}\}$. We say that an axiom or set is* below *a set Γ_n if it is either an element of Γ_n, or if it is (recursively) below some element of Γ_n.*

For Fig. 3, the diversified knowledge base is $\{A \sqsubseteq \exists R^{n3}.C^{n3}, \exists R^{n3}.C^{n3} \sqsubseteq B^{n1}, B^{n1} \sqsubseteq C\}$ and we have $\Gamma_{n1} = \{B^{n1} \sqsubseteq C, \{\exists R^{n3}.C^{n3} \sqsubseteq B^{n1}, \{A \sqsubseteq \exists R^{n3}.C^{n3}\}\}\}$. Since the underlying calculus is correct, the conclusion still follows from the diversified knowledge base, and the diversified proof tree is still correct. Below we use diversification to construct proof trees with invalid conclusions for calculi with insufficient arities.

To this end, note that if l is the maximal number of premises in rules of K, then each set Γ_n has at most l elements (axioms α'_m for leaf children, sets Γ_m for non-leaf children). Moreover, if $\Gamma_m \in \Gamma_n$, then the DL signature symbols that occur in axioms below Γ_m either belong to the interface of n, or occur only in axioms of KB′ that are below Γ_m. The interface includes all DL symbols that occur in the ground IDB atom that is derived at a certain node of the proof tree, so the use of auxiliary constants can

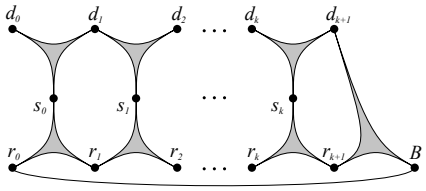

Fig. 4. Dependency graph for the proof of Theorem 4

require the inclusion of *all* symbols of a given input axiom into the interface. Yet, the arity clearly limits the number of axioms for which this may be the case: for a calculus of arity a, the interface of any node can comprise no more than the set of DL symbols that occur in a axioms of the input knowledge base.

These observations can also be interpreted graphically based on the *dependency graph* of KB′ – the graph that has the signature symbols in KB′ as its nodes, and, for each axiom of KB′ with exactly n signature symbols, an n-ary hyperedge connecting these n symbols. The sets of axioms Γ_n can be viewed as subgraphs of a dependency graph, where the interface of the node n describes the nodes that this subgraph is allowed to share with the remaining graph. These insights allow us to provide a proof sketch for our first minimality result; see [6] for details on each step of the argument.

Theorem 4. *Let $\mathcal{L}$ be a DL with GCIs, existential quantification, and role chains. Every materialisation calculus that is sound and complete for classification or instance retrieval in $\mathcal{L}$ has arity three or more.*

Proof. To obtain the result for classification, suppose that there is a binary materialisation calculus $K = \langle I, P, O\rangle$ for classification in $\mathcal{L}$. Let KB contain the following axioms:

$$D_i \sqsubseteq \exists S_i.D_{i+1}, \quad S_i \circ R_{i+1} \sqsubseteq R_i, \quad D_{k+1} \sqsubseteq \exists R_{k+1}.B, \quad \exists R_0.B \sqsubseteq B,$$

for all $i \in \{0, \ldots, k\}$, where $k > 2(l+1)$ for l the maximal number of body atoms in rules of P. Then KB entails $D_0 \sqsubseteq B$. Thus there is a proof tree T for deriving $O(D_0 \sqsubseteq B)$ for the program $I(\text{KB}) \cup I(\mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}) \cup P$. Let $T' = \langle N, E, \lambda'\rangle$ be the diversified proof tree obtained from T by using renamed symbols s^n as in Definition 3, and let KB′ be the according diversified knowledge base. One can now construct a model $\mathcal{I}$ of KB′ in such a way that $\mathcal{I} \models D_0 \sqsubseteq B$ can hold only if KB′ contains axioms of the form:

$$d_0 \sqsubseteq s_0.d_1, \ldots, d_k \sqsubseteq s_k.d_{k+1}, \quad s_0 \circ r_1 \sqsubseteq r_0, \ldots, s_k \circ r_{k+1} \sqsubseteq r_k, \quad d_{k+1} \sqsubseteq B, \ \exists r_0.B \sqsubseteq B,$$

where $d_0 = D_0$, $d_i = D_i^o$ for some $o \in N$, $s_i = S_i^o$ for some $o \in N$, and $r_i = R_i^o$ for some $o \in N$. We claim that this is impossible. For a contradiction, suppose KB′ contains a set of axioms KB″ of this form. The axioms of KB″ are distributed over sets $(\Gamma_o)_{o\in N}$ as in Definition 4. Since T' has an out-degree of at most l (as specified above), our choice of k implies that T' contains a node $o \in N$ such that Γ_o has three axioms of the form $d_i \sqsubseteq \exists s_i.d_{i+1}$ below it, and such that three other axioms of this form are not below it.

The axioms below Γ_o induce a subgraph of the dependency graph of KB″ as shown in Fig. 4. As discussed above, this subgraph may share at most two nodes with the rest of the graph since K has arity two. Now it is not hard to argue that such a subgraph

cannot exist. Hence Γ_o cannot exist, and KB$''$ cannot be contained in KB$'$. So $\mathcal{I}$ does not satisfy $D_0 \sqsubseteq B$, and thus the latter is not a consequence of KB$'$. As T' is a proof tree for $I(\text{KB}') \cup I(\mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}) \cup P$, K derives $D_0 \sqsubseteq B$. So K cannot be sound, contradicting our assumption of its existence.

The result for instance retrieval is obtained by extending KB with an axiom $D_0(a)$, and using an analogous argument to show that $B(a)$ is not entailed by any diversification of this knowledge base on a materialisation calculus of arity 2. □

Analogous proofs can be given to obtain results for DLs that include nominals:

Theorem 5. *Let $\mathcal{L}$ be a DL with GCIs, existential quantification, and nominal classes. Every materialisation calculus that is sound and complete for classification in $\mathcal{L}$ has arity three or more.*

Theorem 6. *Let $\mathcal{L}$ be a DL with GCIs, existential quantification, role chains, and nominal classes. Every materialisation calculus that is sound and complete for classification in $\mathcal{L}$ has arity four or more.*

These results do not extend to instance retrieval, so in a sense classification is harder to implement efficiently. Indeed, Theorem 1 shows that a ternary instance retrieval calculus exists for a DL that includes existentials, nominals, and role chains. For DLs as in Theorem 5, we have not presented calculi of optimal arity. A ternary (binary) calculus for classification (instance retrieval) in this case can be obtained by eliminating the `triple_sc` (`triple`) predicate from K_{sc} (K_{inst}) as done for the binary calculus K_{sc} presented in [6]. Theorem 6 may be surprising, given that the calculus proposed in [2] for $\mathcal{EL}^{++}$ would be ternary in our notation. The explanation is that this algorithm is incomplete for classification; the proof of Theorem 6 can be used to find a suitable counter example [6].

6 Summary and Conclusions

The focus of this work has been the study of inferencing calculi for $\mathcal{SROEL}(\sqcap, \times)$ and its fragments, and especially this paper is – to the best of our knowledge – the first to present a sound and complete polynomial time calculus for inferencing in a DL that is so closely related to the OWL EL ontology language. For investigating properties of such calculi, we presented a simple framework for expressing materialisation calculi in terms of datalog. This revealed the arity of IDB predicates as an interesting measure for the worst-case space requirements of materialisation-based algorithms. While $\mathcal{SROEL}(\sqcap, \times)$ fragments without role chains and nominals admit classification calculi based on binary IDB predicates, the inclusion of either feature increases the required arity by one. Having both features, $\mathcal{SROEL}(\sqcap, \times)$ thus does not admit any sound and complete classification calculus of arity below four.

We are thus able to differentiate various $\mathcal{SROEL}(\sqcap, \times)$ fragments and inferencing tasks based on a measure that relates to the efficiency of actual implementations. Indeed, our findings agree with practical experiences that especially nominals and role chains are harder to implement efficiently than basic $\mathcal{EL}$ features.[3] Computational complexity

[3] Based on the author's experience implementing Orel [7], and personal communication with developers of DB [4] and CEL (`http://lat.inf.tu-dresden.de/systems/cel/`).

has not been able to provide an explanation for such discrepancies, since all reasoning problems we consider are P-complete. In addition, our study also shows that various other features are not harder to implement than some of the most basic ones, thus providing guidance for deciding which features to implement or to use in an application.

Although there are standard implementation strategies for datalog reasoning, our study is independent of actual algorithms. A promising next step thus is to develop control strategies for implementing our calculi in a "pay-as-you-go" algorithm that minimises the potential negative impact of the occurrence of certain features. Moreover, we conjecture that our results about datalog arity can be further strengthened to obtain more direct statements about space complexity of almost arbitrary monotone calculi.

Acknowledgements The author thanks Yevgeny Kazakov for his valuable input, and the anonymous reviewers for helpful comments. This work was supported by DFG in project *ExpreSST* and by EPSRC in project *ConDOR* (EP/G02085X/1).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley (1994)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Kaelbling, L., Saffiotti, A. (eds.) Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). pp. 364–369. Professional Book Center (2005)
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: Clark, K.G., Patel-Schneider, P.F. (eds.) Proc. OWLED 2008 DC Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 496. CEUR-WS.org (2008)
4. Delaitre, V., Kazakov, Y.: Classifying $\mathcal{ELH}$ ontologies in SQL databases. In: Patel-Schneider, P.F., Hoekstra, R. (eds.) Proc. OWLED 2009 Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org (2009)
5. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 57–67. AAAI Press (2006)
6. Krötzsch, M.: Efficient inferencing for the description logic underlying OWL EL. Tech. Rep. 3005, Institute AIFB, Karlsruhe Institute of Technology (2010), available online at http://www.aifb.kit.edu/web/Techreport3005
7. Krötzsch, M., Mehdi, A., Rudolph, S.: Orel: Database-driven reasoning for OWL 2 profiles. In: Haarslev, V., Toman, D., Weddell, G. (eds.) Proc. 23rd Int. Workshop on Description Logics (DL'10) (2010)
8. Krötzsch, M., Rudolph, S., Hitzler, P.: ELP: Tractable rules for OWL 2. In: Sheth et al. [12], pp. 649–664
9. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language: Profiles. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-profiles/
10. Motik, B., Sattler, U.: A comparison of reasoning techniques for querying large description logic ABoxes. In: Hermann, M., Voronkov, A. (eds.) Proc. 13th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'01). LNCS, vol. 4246, pp. 227–241. Springer (2006)
11. Rudolph, S., Krötzsch, M., Hitzler, P.: Description logic reasoning with decision diagrams: Compiling $\mathcal{SHIQ}$ to disjunctive datalog. In: Sheth et al. [12], pp. 435–450
12. Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.): Proc. 7th Int. Semantic Web Conf. (ISWC'08), LNCS, vol. 5318. Springer (2008)