Towards SPARQL-Based Induction for Large-Scale RDF Data sets

Technical Report

Simon Bin[*] Lorenz Bühmann[*] Jens Lehmann[†]
Axel-Cyrille Ngonga Ngomo[*]

[*]Universität Leipzig, Germany, email: {sbin,buehmann,ngonga}@informatik.uni-leipzig.de
[†]University of Bonn and Fraunhofer IAIS, Germany, email: jens.lehmann@cs.uni-bonn.de and jens.lehmann@iais.fraunhofer.de

We show how to convert OWL Class Expressions to SPARQL queries where the instances of that concept are – with restrictions sensible in the considered concept induction scenario – equal to the SPARQL query result. Furthermore, we implement and integrate our converter into the CELOE algorithm (Class Expression Learning for Ontology Engineering). Therein, it replaces the position of a traditional OWL reasoner, which most structured machine learning approaches assume knowledge to be loaded into. This will foster the application of structured machine learning to the Semantic Web, since most data is readily available in triple stores. We provide experimental evidence for the usefulness of the bridge. In particular, we show that we can improve the runtime of machine learning approaches by several orders of magnitude. With these results, we show that machine learning algorithms can now be executed on data on which in-memory reasoners could not be use previously possible.

1 Introduction and Motivation

A growing amount of data from diverse domains is being converted into RDF[1] as demonstrated by the growth of the Linking Open Data Cloud.[2] With this conversion come a significant number of complex applications which rely on large amounts of data in RDF and OWL[3] to perform demanding tasks, such as detecting patients with particular diseases [SPN$^+$13]. In many cases, machine learning on structured data [Leh09] is of central importance to achieve these goals. These learning approaches commonly rely on reasoners to check whether a set of objects in that data belong to some hypothesis concept. Many reasoners have been developed to support this task, of which most load all the data they require into memory, for a list of reasoners accessible via the OWL API interfaces see [4]. While this is a satisfactory approach for small data sets, a large fraction of the RDF data sets on the Web do not fit into the memory of standard servers. For example, the cancer patient data in LinkedTCGA[5] (20.4 billion facts) [SKI$^+$14] is currently distributed across ten endpoints to ensure that it can answer SPARQL queries in acceptable runtimes. Given that the Data Web keeps on growing, it becomes indispensable to develop structured machine learning approaches that can scale up to very large data sets.

The backbone for scalable structured machine learning would be a means to query large data sets using queries that abide a given OWL entailment regime. Note, that while OWL is inherently intractable, working with tractable subsets can still provide useful results. In this paper, we address the specification of exactly such a backbone for scaling up structured machine learning to large data sets. The intuition behind our approach is as follows: While OWL reasoners can provide the required information, they do not scale to large data sets. On the other hand, the SPARQL query language was developed specifically to query large amounts of data. Hence, by creating a bridge between SPARQL and OWL, we are able to answer OWL queries on large amounts of data. This intuition does not come without challenges. First, full OWL reasoning on very large data sets is inherently difficult due to the time complexity of OWL.[6] However, full OWL reasoning is rarely needed for machine learning. A second challenge lies in the closed-world and unique names assumption underlying SPARQL, which stands in contradiction with the open-world assumption underlying OWL. We argue that, specifically for machine learning, those assumptions are very common. As a simple example, inferring that a car has exactly 4 wheels under the open world assumption would usually require explicating that all 4 wheels in the car are mutually different (i.e. are unique) and that there is no other wheel specified in some other knowledge base (i.e. a closed world). Here, we make use of the sufficiency of the closed-world

[1] `http://www.w3.org/TR/rdf11-concepts/`
[2] `http://lod-cloud.net/`
[3] `http://www.w3.org/TR/owl2-overview/`
[4] `http://owlapi.sourceforge.net/reasoners.html`
[5] `http://aksw.org/Projects/LinkedTCGA`
[6] Complexity of OWL 2 DL (a common subset of OWL) is 2NEXPTIME

assumption for a large number of machine learning tasks. Taking those requirements in a machine learning context into account, we define a formal rewriting procedure mapping OWL instance checks (as the most frequent reasoning procedure in the refinement operator based CELOE machine learning algorithm [LABT11]) to SPARQL queries. We show the formal equivalence of the returned instances with respect to this rewriting procedure, under certain restrictions. This result enables the use of large amounts of existing OWL and RDF data to solve classification problems. We then evaluate this bridge by extending an existing algorithm and performing benchmark machine learning tasks.

The rest of this paper is structured as follows: We begin by summarising the semantics of OWL and SPARQL (Section 2). We then define a rewriting algorithm for transforming OWL expressions into SPARQL (Section 3). This rewriting is then utilised in a machine learning approach for RDF data based on refinement operators (Section 4). Our approach is subsequently evaluated on bio-medical data (Section 5), and put into context of related work (Section 6). Finally, we conclude with a discussion and point out future work (Section 7).

2 Semantics of OWL and SPARQL

2.1 DL Semantics

The semantics of OWL is based on Description Logics. Description Logics is the name of a family of knowledge representation (KR) formalisms. They emerged from earlier KR formalisms like semantic networks and frames. Their origin lies in the work of Brachman on structured inheritance networks [Bra78]. Since then, description logics have enjoyed increasing popularity. They can essentially be understood as fragments of first-order predicate logic. They have less expressive power, but usually decidable inference problems and a user-friendly variable free syntax.

Description logics represent knowledge in terms of *objects*, *concepts*, and *roles*. Concepts formally describe notions in an application domain, e.g. one could define the concept of being a father as "a man having a child" ($\texttt{Father} \equiv \texttt{Man} \sqcap \exists \texttt{hasChild}.\top$ in DL notation). Objects are members of concepts in the application domain and roles are binary relations between objects. Objects correspond to constants, concepts to unary predicates, and roles to binary predicates in first-order logic. Moreover, in OWL a concept is usually called class expression, a role refers to a property expression and an object denotes an individual. The main properties that describe individuals are called object and data properties.[1] Object properties link individuals to individuals, whereas data properties contain data values (numbers, text) about individuals.

In description logics, systems information is stored in a *knowledge base*, which can be divided into two parts: *TBox* and *ABox*. The ABox contains *assertions* about objects. It relates objects to concepts and other objects via roles. The TBox describes the *terminology* by relating concepts and roles. For some expressive description logics this clear separation does not exist. Furthermore, the notion of an *RBox*, which contains knowledge about roles, is sometimes used in expressive description logics. We will usually consider those axioms as part of the TBox in this paper.

The semantics of concepts is defined by means of interpretations. Let there be distinct sets of labels for concepts N_C and roles N_R, respectively. An *interpretation* $\mathcal{I}$ consists of a non-empty *interpretation domain* $\Delta^\mathcal{I}$ and an *interpretation function* $\cdot^\mathcal{I}$, which assigns to each $A \in N_C$ a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and to each $r \in N_R$ a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$.

Let C, D be concepts and $\mathcal{T}$ a TBox. *C is subsumed by D*, denoted by $C \sqsubseteq D$, iff for any model $\mathcal{I}$ we have $C^\mathcal{I} \subseteq D^\mathcal{I}$. *$C$ is subsumed by D with respect to $\mathcal{T}$*, denoted by $C \sqsubseteq_\mathcal{T} D$, iff for any model $\mathcal{I}$ of $\mathcal{T}$ we have $C^\mathcal{I} \subseteq D^\mathcal{I}$.

C is equivalent to D (with respect to $\mathcal{T}$), denoted by $C \equiv D$ ($C \equiv_\mathcal{T} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_\mathcal{T} D$) and $D \sqsubseteq C$ ($D \sqsubseteq_\mathcal{T} C$).

C is strictly subsumed by D (with respect to $\mathcal{T}$), denoted by $C \sqsubset D$ ($C \sqsubset_\mathcal{T} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_\mathcal{T} D$) and not $C \equiv D$ ($C \equiv_\mathcal{T} D$).

[1] `https://www.w3.org/TR/owl-ref/#Property`

Table 2.1: Syntax and semantics for concepts in $\mathcal{SROQ}$.

construct	syntax	semantics
atomic concept	A	$A^\mathcal{I} \subseteq \Delta^\mathcal{I}$
role	r	$r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$
nominals	$\{o\}$	$\{o\}^\mathcal{I} \subseteq \Delta^\mathcal{I}, \|\{o\}\|^\mathcal{I} = 1$
top concept	$\top$	$\Delta^\mathcal{I}$
bottom concept	$\bot$	$\varnothing$
conjunction	$C \sqcap D$	$(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$
disjunction	$C \sqcup D$	$(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$
negation	$\neg C$	$(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$
exists restriction	$\exists r.C$	$(\exists r.C)^\mathcal{I} = \{a \mid \exists b.(a,b) \in r^\mathcal{I}$ and $b \in C^\mathcal{I}\}$
value restriction	$\forall r.C$	$(\forall r.C)^\mathcal{I} = \{a \mid \forall b.(a,b) \in r^\mathcal{I}$ implies $b \in C^\mathcal{I}\}$
atleast restriction	$\geq n\ r.C$	$(\geq n\ r)^\mathcal{I} = \{a \mid \|(\{b \mid (a,b) \in r^\mathcal{I}\}\| \geq n\}$
atmost restriction	$\leq n\ r.C$	$(\leq n\ r)^\mathcal{I} = \{a \mid \|(\{b \mid (a,b) \in r^\mathcal{I}\}\| \leq n\}$

Let $\mathcal{A}$ be an ABox, $\mathcal{T}$ a TBox, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a knowledge base, C a concept, and $a \in N_I$ an object. *a is an instance of C with respect to $\mathcal{A}$*, denoted by $\mathcal{A} \models C(a)$, iff in any model $\mathcal{I}$ of $\mathcal{A}$ we have $a^\mathcal{I} \in C^\mathcal{I}$. *$a$ is an instance of C with respect to $\mathcal{K}$*, denoted by $\mathcal{K} \models C(a)$, iff in any model $\mathcal{I}$ of $\mathcal{K}$ we have $a^\mathcal{I} \in C^\mathcal{I}$.

To denote that a is not an instance of C with respect to $\mathcal{A}$ ($\mathcal{K}$) we write $\mathcal{A} \not\models C(a)$ ($\mathcal{K} \not\models C(a)$).

Let $\mathcal{A}$ be an ABox, $\mathcal{T}$ a TBox, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a knowledge base, C a concept. The *retrieval $R_\mathcal{A}(C)$ of a concept C with respect to $\mathcal{A}$* is the set of all instances of C: $R_\mathcal{A}(C) = \{a \mid a \in N_I$ and $\mathcal{A} \models C(a)\}$. Similarly the *retrieval $R_\mathcal{A}(C)$ of a concept C with respect to $\mathcal{K}$* is $R_\mathcal{K}(C) = \{a \mid a \in N_I$ and $\mathcal{K} \models C(a)\}$.

Table 2.1 lists common DL concepts and their semantics. $\mathcal{SROIQ}$ is a well-known description language, as it is the basis for OWL 2. The letters describe certain features of the language, such as $\mathcal{S}$ for negation, intersection, universal and existential restriction, $\mathcal{R}$ for complex roles, $\mathcal{O}$ for nominals (such as hasValue in OWL), $\mathcal{I}$ for inverse properties, $\mathcal{Q}$ for qualified cardinality restrictions [HKS06]. $\mathcal{SROQ}$ is the subset lacking inverse properties.

2.2 SPARQL Semantics

We use the SPARQL algebra syntax and semantics in [AGP10] in this section and repeat the essential notions below.[2] For a complete definition of the SPARQL syntax and semantics, however, we refer to [AGP10], [PAG09] and the official W3C recommendation[3].

We use V to denote SPARQL variables. A mapping μ from V to a set U is a partial function $\mu : V \to U$. The domain of μ, denoted by $dom(\mu)$, is the subset of V where μ is defined. $var(P)$ is the set of variables contained in a basic graph pattern P. Given a triple pattern t and a mapping μ with $var(t) \subseteq dom(\mu)$, $\mu(t)$ is the triple pattern obtained by replacing the variables in t according to μ. This can be extended to a basic graph pattern P by defining $\mu(P) = \cup_{t \in P}\{\mu(t)\}$.

The evaluation of a basic graph pattern P, denoted by $[[P]]$, in $\mathcal{A}$ is then defined as the set of mappings $[[P]]^{\mathcal{A}} = \{\mu : V \to U \mid (dom(\mu) = var(P)$ and $\mu(P) \subseteq \mathcal{A}\}$. This can be extended to graph patterns and filters. Note that we will sometimes drop $\mathcal{A}$ if it is clear from the context. In the following, we will only introduce the notions required in our proof. For a conjunction of several triple patterns, their semantics are defined as $[[(P_1 \text{ AND } \dots \text{ AND } P_n)]] = [[P_1]] \bowtie \cdots \bowtie [[P_n]]$. The join operator for two sets of mappings Ω_1 and Ω_2 is defined as $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2$ and μ_1, μ_2 are compatible mappings $\}$. Two mappings μ_1 and μ_2 are *compatible* if for all $x \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$. The semantics of UNION patterns are defined as $[[P_1 \text{ UNION } \dots \text{ UNION } P_n]] = [[P_1]] \cup \cdots \cup [[P_n]]$. For filters, we need to introduce the notion of *satisfaction*. Given a mapping μ and a built-in condition R, the satisfaction of R by μ is denoted by $\mu \models R$ in a three valued logic (`true`, `false`, `error`). In our case, built-in conditions of the form `?x = ?y` are relevant, which result in an error if $?x \notin dom(\mu)$ or $?y \notin dom(\mu)$. They are `true` if $\mu(?x) = \mu(?y)$ and `false` otherwise. The built-in construct of the form `?v IN S` where S is a set of expressions, is satisfied if $\mu(?v) \in S$ and `false` otherwise. NOT EXISTS constructs in filters take a graph pattern P as input and evaluate it. They result in `false` if $\mu \in [[P]]$ and `true` otherwise. The semantics of a filter in $\mathcal{A}$ is defined as $[[P \text{ FILTER } R]]^{\mathcal{A}} = \{\mu \in [[P]]^{\mathcal{A}} \mid \mu \models R\}$.

A SELECT query restricts the evaluation of a basic graph pattern to a set of variables W. The evaluation of a SELECT query (W, P) over a data set $\mathcal{A}$ is the set of mappings $[[(W, P)]]^{\mathcal{A}} = \{\mu_{|W} \mid \mu \in [[P]]^{\mathcal{A}}\}$. Since the interpretation of a DL concept results in a subset of the domain of the interpretation, we need to view such an evaluation as a set. Assuming a set Ω of mappings is given, this can be achieved by performing a selection over a single variable v and collecting the results in a set, i.e. $\Omega(v) = \{\mu(v) \mid \mu \in \Omega\}$.

[2]We slightly simplify the notation by dropping the graphs used as we are only working on a single graph here.
[3]`http://www.w3.org/TR/sparql11-query/`

3 OWL Class Expression Rewriting Algorithm

We show the following: Instances of a complex concept are, under certain restrictions, equivalent to the results returned by a SPARQL query obtained as the result of applying the rewriting procedure in Tables 3.1 and 3.2 to the concept. Instance checks are required in machine learning algorithms to verify whether an input example, corresponding to an instance in the knowledge base, is covered by a hypothesis. We believe that our rewriting based on canonical model checking is particular suitable for the considered learning setting in which local closed world settings are preferred. Similar results could be achieved indirectly by first performing Datalog rewriting and then taking advantage of SPARQL and SQL both being based on relational algebra. However, a direct rewriting is naturally more efficient and has the advantage of operating directly on triple stores.

Let $C, D, C_1, \dots, C_n$ be class expressions, A an atomic class, r an object property, $a_1, \dots, a_n$ individuals and $\Theta = \{\leq, \geq, =\}$. Let $\tau(C, v)$ be a mapping from a class expression C and a target variable v to a SPARQL graph pattern $\mathfrak{p}$ as described in Table 3.1. A given axiom pattern $C \sqsubseteq D$ can be converted into a SPARQL query pattern $\mathfrak{p}$ by applying $\mathfrak{p} := \tau(C \sqcap D)$. We assume that τ generates unused query variables in its recursion.

Following the conversion tables, the conversion is straightforward by applying the matching entries in the tables. For example, to convert the class expression "Compound ⊓ ∃ hasStructure.Ar_halide" to a SPARQL WHERE clause:

`{ ?var0 a <Compound>.`	Rule "A" in the table
`?var0 <hasStructure> ?var1.`	Rule "∃r.C"
`?var1 a <Ar_halide>. }`	Rule "A" applied to $\tau(Ar\_halide,$ `?var1`$)$

In the following, we assume that an ABox $\mathcal{A}$ is given, which contains statements of the form $A(a)$, i.e. class assertions to named classes, and $r(a, b)$, i.e. role assertions. In RDF, those facts correspond to triples `a rdf:type C` and `a r b` respectively. We denote the objects, classes and roles in the ABox with N_O, N_C and N_R. Using this, we can define an interpretation $\mathcal{I}$ as:

$$\begin{aligned}
\Delta^{\mathcal{I}} &= N_O, \\
a^{\mathcal{I}} &= a, \\
A^{\mathcal{I}} &= \{a \mid A(a) \in \mathcal{A}\} \quad \text{for all } A \in N_C, \\
r^{\mathcal{I}} &= \{(a, b) \mid r(a, b) \in \mathcal{A}\} \quad \text{for all } r \in N_R.
\end{aligned}$$

Table 3.1: Conversion of class expressions into a SPARQL graph pattern.

Class Expression C_i	Graph Pattern $\mathfrak{p} = \tau(C_i, \texttt{?var})$
A	`{?var rdf:type A.}`
$\neg C$	`{?var ?p ?o . FILTER NOT EXISTS {` $\tau(C, ?var)$ `}}`
$\{a_1, \ldots, a_n\}$	`{?var ?p ?o . FILTER (?var IN (`$a_1, \ldots, a_n$`))}`
$C_1 \sqcap \ldots \sqcap C_n$	$\{\tau(C_1, ?var) \cup \ldots \cup \tau(C_n, ?var)\}$
$C_1 \sqcup \ldots \sqcup C_n$	$\{\tau(C_1, ?var)\}$ `UNION` $\ldots$ `UNION` $\{\tau(C_n, ?var)\}$
$\exists\, r.C$	`{?var r ?s.}` $\cup\, \tau(C, ?s)$
$\exists\, r.\{a\}$	`{?var r a.}`
$\exists\, r.SELF$	`{?var r ?var.}`
$\forall\, r.C$	`{ ?var r ?s0. { SELECT ?var (count(?s1) AS ?cnt1) WHERE { ?var r ?s1 .` $\tau(C, ?s1)$ `} GROUP BY ?var } { SELECT ?var (count(?s2) AS ?cnt2) WHERE { ?var r ?s2 } GROUP BY ?var } FILTER ( ?cnt1 = ?cnt2 ) }`
$\Theta n\; r.C$	`{ ?var r ?s0. { SELECT ?var WHERE { ?var r ?s .` $\tau(C, ?s)$ `} GROUP BY ?var HAVING ( count(?s)` $\Theta\, n$ `) } }`

Table 3.2: Conversion of property expressions into a SPARQL g.p.

Property Expression p_i	Graph Pattern $\mathfrak{p} = \tau(p_i, \texttt{?var})$
p	`{?var p ?o.}`
p^{-1}	`{?s p ?var .}`
$p_1 \circ \cdots \circ p_n$	`{?var` p_1 `?o1.` $\vdots$ $\vdots$ $\vdots$ `?o_n-1` p_n `?o_n.}`

This interpretation is called the *canonical interpretation* of $\mathcal{A}$ [BCM+03b]. Now let τ be the function defined in Tables 3.1 and 3.2. For simplicity, we assume that τ returns a SPARQL query algebra expression (in contrast to a query string). We can then show that executing the query converted from a concept C using τ over $\mathcal{A}$ is the same as the canonical interpretation of C.

Proposition. *Let $\mathcal{A}$ be an ABox, C a $\mathcal{SROQ}$ concept and $\mathcal{I}$ the canonical interpretation of $\mathcal{A}$. Then the following holds.*

$$C^{\mathcal{I}} = [[\tau(C, ?var)]]^{\mathcal{A}}(?var)$$

Proof. We prove by induction over the structure of class expressions. In each case, we show that the DL and SPARQL semantics are equal with respect to $\cdot^{\mathcal{I}}$ and τ. For the cases $\forall r.C$ and $\Theta n\ r.C$, the conversion between both semantics is only sketched (due to length).

Induction Base. $A^{\mathcal{I}}$ contains the explicit instances of A by the definition of canonical interpretation and $[[\tau((?var, \texttt{rdf:type}, A))]](?var)$ contains the subjects of triples asserting instances to A. Hence, $A^{\mathcal{I}} = [[\tau(A, ?var)]](?var)$.

Induction Step

$C = C_1 \sqcap \cdots \sqcap C_n$: We have $C^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \cdots \cap C_n^{\mathcal{I}}$, i.e. $C^{\mathcal{I}}$ is the intersection of the interpretation of all C_i. By SPARQL semantics, we also have $[[\tau(C, ?var)]] = [[C_1]] \bowtie \cdots \bowtie [[C_n]]$. For all i with $1 \leq i \leq n$, $C_i^{\mathcal{I}} = [[\tau(C_i, ?var)]](?var)$ follows by induction. Since all $\tau(C_i, ?var)$ just contain a single common variable, the compatibility check in the join operation ensures that ?var is mapped to the same entity for the evaluation of each sub-pattern $[[\tau(C_i, ?var)]]$. Thus, the result only contains elements which are in each set $C_i^{\mathcal{I}}$ and, therefore, $(C_1 \sqcap \cdots \sqcap C_n)^{\mathcal{I}} = [[\tau(C_1 \sqcap \cdots \sqcap C_n, ?var)]](?var)$.

$C = C_1 \sqcup \cdots \sqcup C_n$: We have $C^{\mathcal{I}} = C_1^{\mathcal{I}} \cup \cdots \cup C_n^{\mathcal{I}}$, i.e. $C^{\mathcal{I}}$ is the union of the interpretation of all C_i $(1 \leq i \leq n)$. For the SPARQL rewrite, we have $[[\tau(C, ?var)]] = [[\tau(C_1 \text{ UNION } \ldots \text{ UNION } C_n, ?var)]] = [[\tau(C_1, ?var)]] \cup \cdots \cup [[\tau(C_n, ?var)]]$, which is trivially equal to $C^{\mathcal{I}}$ by induction.

$C = \neg C'$: We have $C^{\mathcal{I}} = \Delta \setminus C'^{\mathcal{I}}$ by DL semantics. For SPARQL, we have $[[\tau(C, ?var)]] = \{\mu \in [[(?var, ?p, ?o)]] \mid \mu \notin [[\tau(C', ?var)]]\}$ by filter semantics. In this formula, $[[(?var, ?p, ?o)]]$ contains mappings from `?var` to all elements of the domain Δ, whereas $[[\tau(C', ?var)]](?var) = C'^{\mathcal{I}}$ by induction. Thus, $[[\tau(C, ?var)]] = \Delta \setminus C'^{\mathcal{I}} = C^{\mathcal{I}}$.

$C = \exists r.C'$: $C^{\mathcal{I}} = \{a \mid \exists b : \langle a, b\rangle \in r^{\mathcal{I}} \wedge b \in C'^{\mathcal{I}}\}$ according to DL semantics. This can also be written as a join of a binary relation r and a unary relation C on the second argument of r (and a projection to the first argument of r), i.e. $C^{\mathcal{I}} = \pi_1(r^{\mathcal{I}} \bowtie_{2=1} C'^{\mathcal{I}})$. For SPARQL, we can evaluate $[[\tau(C, ?var)]] = [[(?var, r, ?s)]] \bowtie [[\tau(C', ?s)]]$, i.e. we have the same structure. $[[(?var, r, ?s)]]$ corresponds to $r^{\mathcal{I}}$ and $[[\tau(C', ?s)]]$ corresponds to $C'^{\mathcal{I}}$ by induction. Thus, we get $(\exists r.C')^{\mathcal{I}} = [[\tau(C, ?var)]](?var)$. We do not treat $\exists r.\{a\}$ in the definition of τ separately as it only provides a shortcut to a longer equivalent query.

$C = \{a_1, \ldots, a_n\}$: According to DL semantics, we have $C^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$. For SPARQL, we have $[[\tau(C, ?var)]] = \{\mu \in [[(?var, ?p, ?o)]] \mid \mu(?var) \in C^{\mathcal{I}}\}$ by filter semantics. Similar to negation, $[[(?var, ?p, ?o)]]$ contains mappings from `?var` to all elements of the domain Δ (and is only used as a dummy since filters cannot stand alone), whereas $\mu(?var) \in C^{\mathcal{I}}$ restricts those exactly to the elements of $C^{\mathcal{I}}$.

$C = \forall r.C'$ **(sketch)** : By DL semantics, we have $(\forall r.C)^{\mathcal{I}} = \{a \mid \forall b : \langle a, b\rangle \in r^{\mathcal{I}}$ implies $b \in C'^{\mathcal{I}}\}$, i.e. for all fillers of r, we need to check whether they are in C'. In $\tau(C, ?var)$, we achieve this by first counting the number of all fillers of r in a sub-select (`?cnt1`) and then counting the number of all fillers of r belonging to C' (`?cnt2`). This is done for each subject of r (`GROUP BY ?var`). All subjects, for which both counts are equal, are part of $[[\tau(C, ?var)]]$, i.e. the SPARQL query corresponds exactly to DL semantics.

$C = \Theta n\ r.C$ **(sketch)** : By DL semantics, we have $(\Theta n\ r)^{\mathcal{I}} = \{a \mid |\{b \mid (a, b) \in r^{\mathcal{I}}\}|\ \Theta\ n\}$. Again, the SPARQL query is corresponding exactly to DL semantics: In the SPARQL query, we count the fillers of r (`count(?s)`) grouped by subject (`GROUP BY ?var`). The `HAVING` expression is then used to verify whether the cardinality restriction is satisfied.

□

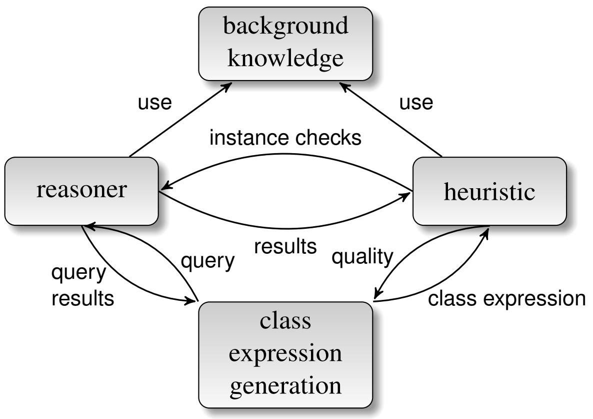


Figure 3.1: Outline of the general learning approach: Class expressions taking the available background knowledge into account are generated and evaluated in a heuristic with respect to the target learning problem. Figure adapted from [HLA09].

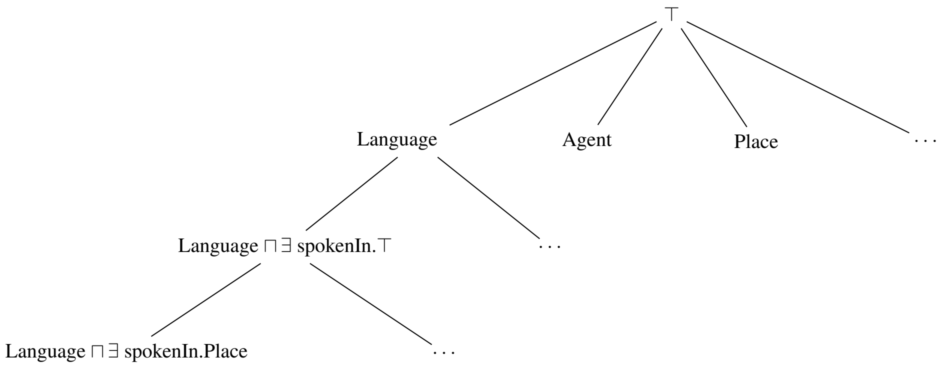


Figure 3.2: Illustration of a simple search tree demonstrating a top-down refinement strategy for learning description logic concepts.

4 Learning Problem and Algorithm

4.1 Learning Problem

The task we investigate resembles *Inductive Logic Programming* [NCdW97] using a description logic knowledge base as background knowledge and OWL / Description Logics as target language. Specifically, we consider supervised machine learning from positive and negative examples, which are resources in a background knowledge base $\mathcal{K}$. To define the class learning problem, we need the notion of a *retrieval* reasoner operation $R_{\mathcal{K}}(C)$, which returns the set of all instances of C in a knowledge base $\mathcal{K}$.

Figure 3.1 gives a brief overview of our base algorithm *CELOE*—Class Expression Learning for Ontology Engineering, which follows the common "generate and test" approach in ILP. This means that learning is seen as a search process and several class expressions are generated and tested against a background knowledge base. Each of those class expressions is evaluated using the heuristic in [LABT11].

4.2 Refinement Operator Based Learning

Refinement operators can be used for searching in the space of complex concepts and provide solutions for learning problems. The operator must induce a quasi-ordering on the target language. As ordering we can use subsumption ($\sqsubseteq$), which is such a quasi-ordering relation. If an expression C subsumes an expression D ($D \sqsubseteq C$), then C will cover all examples which are covered by D. This makes subsumption a suitable order for searching in expressions as it allows to prune parts of the search space without losing possible solutions.

Definition 1 (refinement operator)**.** *A* quasi-ordering *is a reflexive and transitive relation. In a quasi-ordered space* $(S, \preceq)$ *a* downward (upward) refinement operator ρ *is a mapping from* S *to* 2^S*, such that for any* $C \in S$ *we have that* $C' \in \rho(C)$ *implies* $C' \preceq C$ *(*$C \preceq C'$*).* C' *is called a* specialisation *(*generalisation*) of* C.

CELOE relies on a top-down algorithm based on refinement operators as illustrated in Figure 3.2. This means that the first class expression which will be tested is the most general expression ($\top$), which is then mapped to a set of more specific expressions by means of a downward refinement operator. Naturally, the refinement operator can be applied to the obtained expressions again, thereby spanning a *search tree*. The search tree can be pruned when an expression does not cover sufficiently many instances of the class A we want to describe. This can be done because the downward refinement operator guarantees that all refinements of this concept will cover at most equally many instances of A.

One example for a path in a search tree spanned up by a refinement operator is the following ($\rightsquigarrow$ denotes a refinement step):

$$\top \rightsquigarrow \texttt{Language} \rightsquigarrow \texttt{Language}\sqcap\exists\,\texttt{spokenIn}.\top \rightsquigarrow \texttt{Language}\sqcap\exists\,\texttt{spokenIn.Place}$$

The heart of such a learning strategy is to define a suitable refinement operator and an appropriate search heuristic for deciding which nodes in the search tree to expand. We use the refinement operator defined in [LH10] and partially displayed in Figure 4.1. We integrate this operator into a top down refinement based learning strategy sketched in Algorithm 1. Essentially, starting from the $\top$ concept, the algorithm iteratively applies the refinement operator which results in a search tree. In each iteration, the highest scoring node in the search tree is selected and expanded (heuristic search).

We now define the operator and prove that it is indeed a downward refinement operator. For each $A \in N_C$, we define:

$$\begin{aligned} \textit{sh}_{\downarrow}(A) = \{A' \in N_C \mid A' \sqsubset A, & \\ & \text{there is no } A'' \in N_C \text{ with } A' \sqsubset_{\mathcal{T}} A'' \sqsubset_{\mathcal{T}} A\} \end{aligned} \quad (4.1)$$

$\textit{sh}_{\downarrow}(\top)$ is defined analogously for $\top$ instead of A. $\textit{sh}_{\uparrow}(A)$ is defined analogously for going upward in the subsumption hierarchy. (sh stands for subsumption hierarchy.)

We do the same for roles, i.e. :

$$\begin{aligned} \textit{sh}_{\downarrow}(r) = \{r' \mid r' \in N_R, r' \sqsubset r, & \\ & \text{there is no } r'' \in N_R \text{ with } r' \sqsubset_{\mathcal{T}} r'' \sqsubset_{\mathcal{T}} r\} \end{aligned} \quad (4.2)$$

domain(r) denotes the domain of a role r and *range*(r) the range of a role r. A range axiom links a role to a concept. It asserts that the role fillers must be instances of a given concept. Analogously, domain axioms restrict the first argument of role assertions to a concept.

We define:

$$\begin{aligned} \textit{ad}(r) = \ & \text{an } A \text{ with } A \in \{\top\} \cup N_C \text{ and } \textit{domain}(r) \sqsubseteq A \\ & \text{and there does not exist an } A' \text{ with } \textit{domain}(r) \\ & \sqsubseteq A' \sqsubset A \end{aligned}$$

ar(r) is defined analogously using *range* instead of *domain*. *ad* stands for atomic domain and *ar* stands for atomic range. We assign exactly one atomic concept as domain/range of a role. Since using atomic concepts as domain and range is very common, *domain* and *ad* as well as *range* and *ar* will usually coincide.

The set $\textit{app}_B$ of applicable properties with respect to an atomic concept B is defined as:

$$\textit{app}_B = \{r | r \in N_R, \textit{ad}(r) = A, A \sqcap B \not\equiv \bot\}$$

To give an example, for the concept `Person`, we have that the role `hasChild` with $\textit{ad}(\texttt{hasChild}) = \texttt{Person}$ is applicable, whereas the role `hasAtom` with $\textit{ad}(\texttt{hasAtom}) =$

`ChemicalCompound` is not applicable (assuming `Person` and `ChemicalCompound` are disjoint). We will use this to restrict the search space by ruling out unsatisfiable concepts.

The set of most general applicable roles mgr_B with respect to a concept B is defined as:

$$\mathit{mgr}_B = \{r \mid r \in \mathit{app}_B, \text{ there is no } r' \text{ with } r \sqsubset r', r' \in \mathit{app}_B\}$$

M_B with $B \in \{\top\} \cup N_C$ is defined as the union of the following sets:

- $\{A \mid A \in N_C, A \sqcap B \not\equiv \bot, A \sqcap B \not\equiv B,$ there is no $A' \in N_C$ with $A \sqsubset A'\}$
- $\{\neg A \mid A \in N_C, \neg A \sqcap B \not\equiv \bot, \neg A \sqcap B \not\equiv B,$ there is no $A' \in N_C$ with $A' \sqsubset A\}$
- $\{\exists r.\top \mid r \in \mathit{mgr}_B\}$
- $\{\forall r.\top \mid r \in \mathit{mgr}_B\}$

M_B is the set of refinements of the $\top$ concept excluding the use of disjunction ($\sqcup$) and restricted to those which are not disjoint with B.

The operator ρ is defined in Figure 4.1. Note that ρ delegates to an operator ρ_B with $B = \top$ initially, where B is set to the atomic range of roles contained in the input concept when the operator recursively traverses the structure of the concept. The index B in the operator (and the set M above) is used to rule out concepts which are disjoint with B.

We use the following notions for different kinds of refinement steps in ρ:

1. $\overset{\sqcap}{\leadsto}$: add an element conjunctively (cases 3, 4, 5, 6, 8 in the definition of ρ_B in Figure 4.1)
2. $\overset{\top}{\leadsto}$: refine the top concept (case 2 in the definition of ρ_B in Figure 4.1)
3. $\overset{A}{\leadsto}$: refine an atomic concept (case 3 in the definition of ρ_B in Figure 4.1)
4. $\overset{\neg A}{\leadsto}$: refine a negated atomic concept (case 4 in the definition of ρ_B in Figure 4.1)
5. $\overset{r}{\leadsto}$: refine a role (cases 5, 6 in the definition of ρ_B in Figure 4.1)

In the context of the proposed work, the most relevant aspects are the reasoning operations required to run such an algorithm. For constructing refinements, the operator issues domain, range and disjointness reasoner queries. The domains and ranges have to be facts in the knowledge base that is being used.

Disjointness can, for the SPARQL approach, even be expressed directly as a single SPARQL query using a SPARQL `FILTER`, whereas it has to query the list of instances individually for each class and calculate the disjointness manually for the generic OWL reasoner case. Those queries are needed for emptiness checks as well as getting appropriate filler concepts of existential and universal restriction.

The most time-critical operation are instance checks using a closed world assumption. As explained in Section 3, the DL concepts can be converted into SPARQL queries performing model checking instead of full OWL reasoning. In contrast to the interaction with OWL reasoners, where the reasoner is queried for each instance whether it belongs to a concept, this can be expressed in

$$\rho(C) = \begin{cases} \{\bot\} \cup \rho_\top(C) & \text{if } C = \top \\ \rho_\top(C) & \text{otherwise} \end{cases}$$

$$\rho_B(C) = \begin{cases} \varnothing & \text{if } C = \bot \\ \{C_1 \sqcup \dots \sqcup C_n \mid C_i \in M_B \ (1 \leq i \leq n)\} & \text{if } C = \top \\ \{A' \mid A' \in sh_\downarrow(A)\} & \text{if } C = A \ (A \in N_C) \\ \quad \cup \{A \sqcap D \mid D \in \rho_B(\top)\} & \\ \{\neg A' \mid A' \in sh_\uparrow(A)\} & \text{if } C = \neg A \ (A \in N_C) \\ \quad \cup \{\neg A \sqcap D \mid D \in \rho_B(\top)\} & \\ \{\exists r.E \mid A = ar(r), E \in \rho_A(D)\} & \text{if } C = \exists r.D \\ \quad \cup \{\exists r.D \sqcap E \mid E \in \rho_B(\top)\} & \\ \quad \cup \{\exists s.D \mid s \in sh_\downarrow(r)\} & \\ \{\forall r.E \mid A = ar(r), E \in \rho_A(D)\} & \text{if } C = \forall r.D \\ \quad \cup \{\forall r.D \sqcap E \mid E \in \rho_B(\top)\} & \\ \quad \cup \{\forall r.\bot \mid & \\ \qquad D = A \in N_C \text{ and } sh_\downarrow(A) = \varnothing\} & \\ \quad \cup \{\forall s.D \mid s \in sh_\downarrow(r)\} & \\ \{C_1 \sqcap \dots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \dots \sqcap C_n \mid & \text{if } C = C_1 \sqcap \dots \sqcap C_n \\ \qquad D \in \rho_B(C_i), 1 \leq i \leq n\} & (n \geq 2) \\ \{C_1 \sqcup \dots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \dots \sqcup C_n \mid & \text{if } C = C_1 \sqcup \dots \sqcup C_n \\ \qquad D \in \rho_B(C_i), 1 \leq i \leq n\} & (n \geq 2) \\ \quad \cup \{(C_1 \sqcup \dots \sqcup C_n) \sqcap D \mid & \\ \qquad D \in \rho_B(\top)\} & \end{cases}$$

Figure 4.1: Definition of the refinement operator ρ. For details and definitions of $sh_\downarrow$, $sh_\uparrow$, *ar*, M_B we refer to [LH10]

a single SPARQL query selecting all instances of a concept and intersecting it with the list of instances to check.

In the future we also aim to avoid including lists of instances in the query by adding this information to the SPARQL endpoint directly.

4.3 Learning Algorithm

Learning concepts in DLs is a search process. In our proposed learning algorithm, the refinement operator $\rho^{cl}_\downarrow$ is used for building the search tree, while a heuristic decides which nodes to expand. As mentioned before, we want to tackle the infinity of the operator by considering only refinements up to some length n at a given time. We call n the *horizontal expansion* of a node in the search

Algorithm 1: Learning algorithm sketch for refinement-operator based top-down learning with iterative length expansion in search tree nodes (see [LABT11, LH10] for details).

```
   Input: background knowledge, examples $E$, noise in [0,1]
1  $ST$ (search tree) is set to the tree consisting only of the root node $\top$;
2  while $ST$ does not meet the termination criterion do
3      choose a node $N$ with highest score in $ST$
4      expand $N$ up to length $n + 1$, i.e. begin
5          compute list of direct refinements on $N$
6          transform refinements to negation normal form
7          check redundancy of refinements
8          evaluate non-redundant nodes
9          permanently store node expansion
10 Return found concepts in $ST$
```

tree. It is a node specific upper bound on the length of child concepts, which can be increased dynamically by the algorithm during the learning process.

To deal with this, we formally define a *node* in a search tree to be a triple (C, n, b), where C is a concept, $n \in \mathbb{N}$ is the horizontal expansion, and $b \in \{\text{true}, \text{false}\}$ is a boolean marker for the redundancy of a node.

To define a search heuristic for our learning algorithm, we need some notions to be able to express what we consider a good node for expansion. Similarly to existing ILP systems, we use the learning algorithm parameter *noise*, bounding the minimum acceptable training set accuracy of the learned definition. $(1 - \textit{noise})$ is the lowest accuracy a concept needs to have to be considered a solution of a learning problem.

The search heuristics selects the node with the highest score in the search tree at a given time, where the score of a node is defined as follows:

Definition 2 (score). *Let $N = (C, n, b)$ be a node and `Target` the target concept. We introduce the following notions:*

$$\begin{aligned}
\textit{accuracy}(C) &= 1 - \frac{\textit{up} + \textit{cn}}{|E|} \\
\textit{acc\_gain}(N) &= \textit{accuracy}(C) - \textit{accuracy}(C') \\
&\qquad \textit{where } C' \textit{ is the concept in the parent of } N \\
&\qquad \textit{up} = |\{e \mid e \in E^+, \mathcal{K} \cup \{\texttt{Target} \equiv C\} \not\models e\}| \quad \textit{(uncovered positives)} \\
&\qquad \textit{cn} = |\{e \mid e \in E^-, \mathcal{K} \cup \{\texttt{Target} \equiv C\} \models e\}| \quad \textit{(covered negatives)} \\
&\qquad E = E^+ \cup E^-
\end{aligned}$$

If $\textit{up} > \lfloor \textit{noise} \cdot |E| \rfloor$, then the node is too weak*, i.e. it is not a solution candidate and will never be expanded.*

If the node is not too weak, then its score is defined as follows:

$$\mathit{score}(N) = \mathit{accuracy}(C) + \alpha \cdot \mathit{acc\_gain}(N) - \beta \cdot n \quad (\alpha \geq 0,\ \beta > 0)$$

By default, we choose $\alpha = 0.5$ and $\beta = 0.02$. The heuristic uses accuracy as main criterion. Accuracy gain, controlled by α, is incorporated, because those concepts having led to an improvement in accuracy are more likely to be significant refinements towards a solution. As a third criterion, controlled by β, we bias the search towards shorter concepts and less explored areas of the search space. Using horizontal expansion instead of concept length makes the algorithm more flexible in searching less explored areas of the search space and avoids that it gets stuck on concepts with high accuracy and accuracy gain. The score function can be defined independently of the core learning algorithm.

We have now introduced all necessary notions to specify the complete learning algorithm, given in Algorithm 1. *checkRed* is the redundancy check function and *transform* the function to transform a concept to ordered negation normal form.

As we can see, the learning algorithm performs a top down refinement operator driven heuristic search. The main difference to other learning algorithms of this kind is the replacement of a full node expansion by a one step horizontal expansion and the use of a redundancy check procedure.

Apart from knowledge representation, another difference to many ILP programs is the search strategy: Often, ILP tools perform a clause by clause set covering approach to construct a solution step-wise. In DL-Learner, each concept represents a full solution, which is related to single predicate theory learning [Bra99] in ILP. A benefit is that there is no risk in performing possibly suboptimal choices and it is often possible to learn shorter solutions. However, it also leads to a higher runtime and memory consumption. To counteract this, a divide and conquer strategy as extension of Algorithm 1 can be activated in DL-Learner. It restricts the set of nodes which are candidates for expansion to a set of fixed size in regular time intervals. By default, the candidate set is restricted to the 20 most promising nodes each 300 seconds. Those concepts are selected according to their accuracy with a bias towards short concepts with high accuracy on positive examples, since those concepts are more likely to improve in a downward refinement algorithm. We omit the details of this process for brevity. It can be used as a trade off between performance and the risk to make suboptimal decisions.

Correctness of the algorithm has been shown in [LH10].

Proposition 1 (correctness). *If a learning problem has a solution in $\mathcal{ALC}$, then Algorithm 1 terminates and computes a correct solution of the learning problem.*

5 Evaluation

In this section, we aim to analyse the scalability and accuracy of our approach on 3 benchmark learning tasks. We want to answer the following research questions:

Q1 How does the approach scale? More specifically, how many hypotheses can we test in a given time frame when employing SPARQL for concept evaluation compared to an existing approach using an OWL reasoner?

Q2 Does the use of SPARQL influence the accuracy of the predictions?

Q3 Can we solve problems that were previously computationally too expensive to solve?

Q4 Is it beneficial to materialise inferences first?

To this end, we compare variations of the same algorithm (i.e., with in-memory reasoner and with OWL to SPARQL bridge) implemented in the DL-Learner.

5.1 Experimental Setup

We implemented the SPARQL querying method as an extension to the DL-Learner framework as described in the previous section. We use three data sets to compare the SPARQL induction to OWL reasoners. We selected one small and two medium-size data sets which can be processed by an OWL reasoner and reflect typical classification tasks. All our experiments are binary classification tasks. Specifically, the *Carcinogenesis* and *Mutagenesis* data sets are moderately-sized data sets converted from data provided by the Oxford University Machine Learning group.[1] The carcinogenesis task is described in more detail on the DL-Learner website.[2] The task is to predict which chemical compounds cause cancer.

The mutagenesis data set is based on the results of [DLdCD$^+$91]. Its aim is to predict mutations of organisms and it is generally much less challenging than carcinogenesis.

The *Mammographic Mass data set* (Mammograms) is also related to the prediction of cancer. It was published in [ESWW07] and can also be obtained on the web.[3] This data set is much smaller and has only few classes and properties. Here, the aim is to predict the severity (benign or malignant) of a mammographic mass lesion from BI-RADS attributes and the patient's age.

Additionally, we evaluated our approach on an excerpt from the cancer patient data in LinkedTCGA[4] (35 million triples). This data set has not previously been possible to use with

[1] `http://www.cs.ox.ac.uk/activities/machlearn/applications.html`
[2] `http://dl-learner.org/community/carcinogenesis/`
[3] `http://archive.ics.uci.edu/ml/datasets/Mammographic+Mass`
[4] `http://aksw.org/Projects/LinkedTCGA`

Table 5.1: Data set characteristics.

number of	triples	classes	data/object properties		expressivity
carcinogenesis	74,567	142	15	4	$\mathcal{ALC}(\textsc{d})$
mutagenesis	62,067	86	6	5	$\mathcal{AL}(\textsc{d})$
mammograms	6,809	19	2	3	$\mathcal{AL}(\textsc{d})$
TCGA-A	35,329,868	24	113	48	$\mathcal{AL}(\textsc{d})$

DL-Learner because it refuses to load in any reasonable time into the Pellet, HermiT and FaCT reasoners [TH06] (on our test system).

In Table 5.1, the main characteristics of the data sets are described. The number of RDF triples describes the total size of the data set. Furthermore, the total number of classes and data as well as object properties (across all classes) is indicated. This is one indicator for the complexity of the learning progress, since (unlike data properties) object properties can lead to potentially long nested concepts by describing the linked individuals in terms of their properties (and so on). The expressivity refers to the description logic language features that are used in the data set, $\mathcal{AL}$ stands for atomic negation, intersection, existential and universal restrictions, $\mathcal{C}$ for concept negation, and $(\textsc{d})$ for usage of data types.

The approaches are evaluated using three different access options inside the DL-Learner framework. For OWL Reasoners, the data sets are loaded into the reasoner via the OWL API. We tested the popular HermiT[5] and Pellet OWL reasoners [SPG$^+$08]. For SPARQL, the data was loaded into an in-memory Jena[6] model. Lightweight reasoning based on (incomplete) OWL/Lite inference rules[7] was enabled. For comparison, we also precomputed all the inferences externally, in which case the SPARQL back-end acts as a pure graph database. In our evaluation setup, 22 seconds (mutagenesis), 36 seconds (carcinogenesis), 7 seconds (mammograms) were spent on pre-computing all inferences externally beforehand.

For the LinkedTCGA data set, we loaded it into a SPARQL endpoint (running OpenLink Virtuoso[8] 7.1) as this is the way it is commonly provided on the Web. We set up our own copy using an excerpt of the TCGA data obtained from LargeRDFBench.[9] All experiments were run on an AMD Opteron 6376 @ 2.3GHz with 256 GB system memory, of which the DL-Learner framework used 32 GB. Note that DL-Learner itself is single-threaded at this moment and thus did not make use of the full potential of the CPU.

The evaluation test set-up was as follows: For each data set, we took an existing binary classification problem. Sets of positive and negative example instances were used as the input for the machine learning algorithm. We took the positive-negative input from the training files that are included in the DL-Learner. We then applied the existing CELOE algorithm as described in Section 4 on the data set, given the input examples list and a data set specific configuration of noise

[5]`http://www.hermit-reasoner.com/`
[6]`https://jena.apache.org/`
[7]`https://jena.apache.org/documentation/inference/`
[8]`http://virtuoso.openlinksw.com/`
[9]`https://code.google.com/p/bigrdfbench/`

Table 5.2: Number of concept tests within 4000 seconds (mean average in ten runs).

	SPARQL Precomp.	SPARQL Micro-rule	HermiT	Pellet
carcinogenesis	162,430	60,527	87	93
mutagenesis	11,713	4,552	timeout	176
mammograms	26,036	12,260	28	173

level and start class for the class expression refinement. We needed to define a stopping criterion, for which we set the maximum execution time for the CELOE algorithm to 4000 seconds. Then, the algorithm was executed for both reasoners as well as the two different SPARQL access methods. The configuration used in all experiments was exactly the same except for the data sets. The algorithm executed the refinement step and instance checking, and reported every time a concept has been generated which is more accurate with regard to the input example sets. The configuration files can be found in the DL-Learner repository.[10] The total size of the input sets was as follows: Carcinogenesis (338), Mutagenesis (231), Mammograms (962), TCGA-4 (40) individuals of the respective class (Compounds, Compounds, Patients, Patients).

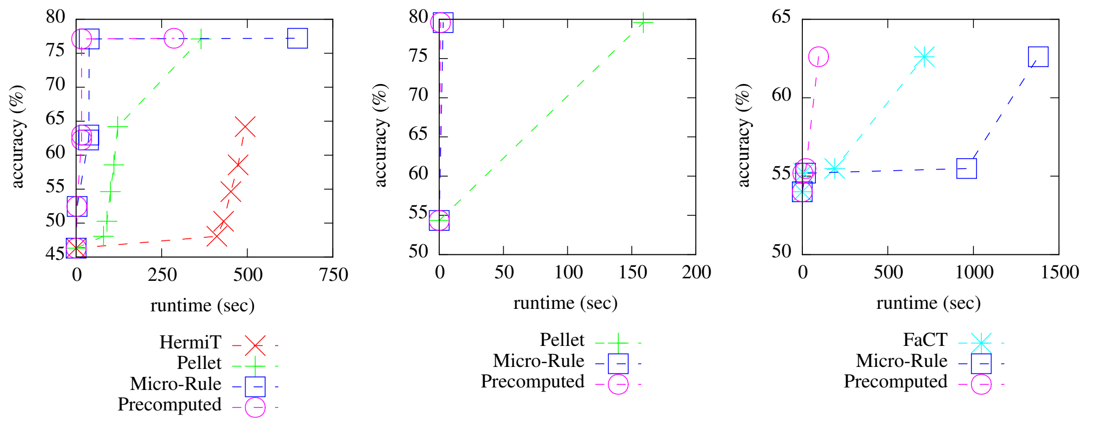
We believed the most distinctive factor would be a larger *number of description tests* that the CELOE algorithm would be able to execute within the demanded time limit. We thus measured this value. Furthermore, we recorded the exact search tree as well as accuracy scores measured through the algorithm. The accuracy was calculated as sum of number of positive and negative input examples that were identified correctly by the class expression divided by the total number of input examples.

5.2 Results

The results of using our SPARQL-based approach to machine learning are very promising. As we can see in Table 5.2, using SPARQL queries exceeds traditional reasoners in the number of concept tests executed. In particular, we can see that they are nearly two orders of magnitude slower with regard to the number of concept tests in the carcinogenesis training example when compared with the in-memory SPARQL access.

We compared the beginnings of the search trees of Pellet and the micro-rule reasoner and verified that both methods are visiting exactly the same nodes. Of course we should keep in mind that the micro-rule reasoner is incomplete, so the entailments may not be as complete as those returned by full OWL reasoners. However, for our data set this was sufficient and played in our favour. For the SPARQL model with materialised inferences, we see that is is again faster than the micro-rule based SPARQL model, which could be expected since now the store only has to perform simple data retrieval. Interestingly, it can be observed that different data sets pose different challenges. While HermiT is generally slower than Pellet in our experiments, the gap is

[10]`https://github.com/AKSW/DL-Learner/tree/sparql-comparison/test/sparql-comparison`



(a) Mammograms (b) Mutagenesis (c) Carcinogenesis

Figure 5.1: Progress of algorithm over time / data set / query method.

smaller on the carcinogenesis benchmark whereas on mutagenesis it did not manage to produce any result in the allotted time frame.

Now if we look at the algorithm progress in Figure 5.1 (in which we compare accuracy versus time), we can see how the greatly increased number of concept tests speeds up the process of locating more accurate class expressions. Out of order, the FaCT reasoner is displayed in comparison for the carcinogenesis benchmark, because both Pellet and HermiT failed to produce a result, and we wanted to show at least one comparison of SPARQL to traditional reasoners. FaCT was not included in the overall comparison because it is missing some features.

Finally, not shown here because we have no baseline to compare to, the TCGA-A benchmark was also decided favourably by the SPARQL based querying. In our test scenario, we managed to solve a simple classification task in 20 minutes. Both HermiT and Pellet did not manage to load the file after 24 hours, when we decided to cancel them.

5.3 Discussion

In this section answer the research questions formulated in Section 5. The answer to our first research question is straight forward. Thanks to the faster operation of the SPARQL instance checking in comparison to that of the OWL reasoners, SPARQL can check more concepts. Since the search algorithm is identical between SPARQL and the OWL reasoners, as long as all required inferences are identical, the result after a number of concept checks will also be identical. Thus, the SPARQL based approach is superior in this regard to the OWL reasoners.

However, we must also consider some additional factors that do not weight in favour of instance checking via SPARQL queries. One is the time needed for materialising the inferences. For all the data sets in this study, this was a non-issue and resolved within seconds. But it is quite probable that this will become much harder as the knowledge base grows in size or the knowledge base contains more elaborate axioms. Similarly, we have not studied the effect on growing knowledge base sizes on the in-memory SPARQL model. Here, it is obvious as well that the model can only be used as long as the knowledge base fits into memory. This limitation however is shared with the OWL reasoners.

By this we have already answered question two as well. Since the search paths are identical, the accuracy is determined in terms of concept checks executed, and thus also identical. As the SPARQL based method is able to run more concept checks within a given time limit compared to the OWL reasoners, the accuracy can only improve. Notwithstanding, no further improvement may be feasible or the improvement may be too far out of reach even for the SPARQL approach using this refinement-based method of concept generation. In our experiments, we could generally observe no worse accuracy from the SPARQL based method than compared to the OWL reasoners.

Thus, it should be discussed in which cases there can be differences. If the ontology requires more complex OWL entailments, the entailment provided by the Jena micro-rule reasoner may not be sufficient. As such, the inferences obtained from OWL reasoners may be different from those in the Jena model and the search may diverge. Ultimately, this question depends on the advance of reasoners inside Graph databases such as Ontotext GraphDB[11], SPARQL 1.1 entailment regimes or further research as how to best implement "reasoning" using SPARQL queries. For example, in [BKPR14] it is shown that OWL QL reasoning may be possible with SPARQL.

Obviously, if the triple store involved does not correctly answer SPARQL queries, this will also lead to diverging results.

For question three, the answer is also a clear yes. Previously, we failed to execute DL-Learner on the LinkedTCGA data set as provided in a SPARQL endpoint. With our new approach, we could successfully execute the learning algorithm on this benchmark problem. We plan to investigate other large data sets once we have identified such data sets and suitable binary classification tasks. Certainly, it is now possible to work on data sets which were previously out of reach.

The final question, is not so clear. When adapted to our test data, precomputing the inferences was always faster than using the micro-rule reasoner. We expect that in general materialisation of inferences should help to improve the execution of machine learning. Since it is possible to trade computation time for storage time, and materialisation means storing the calculated inferences only once, this should be favourable especially when the data sets grow in size, and under the assumption that reasoning is expensive, it only has to be done once. However, it is also possible that not all inferences are needed during the process of searching for a classification. Our current evaluation does not comprise enough data sets to predict in which case materialisation will be better or not.

There do exist hybrid solutions of storing inferences when they are first required. Evaluation of the different materialisation strategies is out of scope for this contribution, however.

In general the SPARQL approach is thus very competitive. However this result must be interpreted carefully. It is not our aim to compete with the feature-set of reasoners. We rather argue, that with the more wide-spread adaption of the Semantic Web and more and more data being available in SPARQL endpoints, a direct way to use these knowledge bases in structured machine learning is required. Our contribution marks the first step towards that goal. Furthermore, we believe that this is in fact required, since it is not feasible to download all the data from those

[11] `http://ontotext.com/products/ontotext-graphdb-owlim/`

endpoints nor to load it all into the reasoner (as has been observed in our TCGA-A experiment). We plan to integrate further existing reasoners or approaches containing entailment steps in the future.

5.4 Summary of Findings

Q1 The experiments indicate that competitive performance of SPARQL based querying can be achieved even for medium sized data sets. The gap will further increase with larger data sets.

Q2 We have observed moderate to large improvements in terms of accuracy for our benchmarks. The actual accuracy is *not* influenced by using SPARQL whatsoever.

Q3 We can now work on data sets which were previously out of reach.

Q4 Materialising inferences had a positive effect on the performance in our benchmarks. However, it depends on the data set, and the time required for materialising.

6 Related Work

Early work on the application of machine learning to Description Logics (DLs) essentially focused on demonstrating the PAC-learnability for various terminological languages. In particular, Cohen and Hirsh investigate the CORECLASSIC DL proving that it is not PAC-learnable [CH92] as well as demonstrating the PAC-learnability of its sub-languages. These approaches tend to cast supervised concept learning to a structural generalising operator working on equivalent graph representations of the concept descriptions. It is also worth mentioning unsupervised learning methodologies for DL concept descriptions, whose prototypical example is KLUSTER [KM94], a polynomial-time algorithm for the induction of BACK terminologies, which exploits the tractability of the standard inferences in this DL language [BCM$^+$03a]. DLs tailored with limited constructors for data modelling in such a way that query answering is not only tractable (i.e. it has very low data complexity inside logarithmic space) but the ABox access can be modelled through SQL have also been studied in [CDGL$^+$05].We are not studying the

Finding a query to describe a set of examples has a long history. The traditional approach of generating a table that can accurately describe the input example sets is described in [Ang87]. Recently, many approaches have been proposed that adopt the idea of *generalisation as search* [Mit82] performed through suitable operators that are specifically designed for DL languages [IPF07, LH09] on the grounds of the previous experience in the context of ILP. There is a body of research around the analysis of such operators [LH10, Leh07] and studies on the practical scalability of algorithms using them [HLA09]. Supervised learning systems, such as YINYANG [IPF07] and DL-Learner [Leh09], which we extended in this work, have been implemented and adoptions implemented for the ontology learning use case [LABT11, BL13]. Also techniques from the area of data mining have been used for unsupervised ontology learning [VN11]. We are using the refinement operator based approach in this contribution, which will take advantage of the class hierarchy encoded in the knowledge base. As an alternative model, a new version of the FOIL algorithm [Qui90] has been implemented, resulting in the DL-FOIL system [FdE08].

A large number of approaches have been developed to improve the scalability of entailment. Many of these approaches have focused on RDFS entailment. For example, a parallel implementation of RDFS entailment over a computer cluster is presented in [WH09]. Their work uses an iterative fix-point approach, applying each RDFS rule to triples until no new inferences can be made. Triples generated by different nodes are written to separate files. Thus, duplicates are not detected and consume both memory bandwidth and storage space. An interesting approach to parallel RDFS reasoning using distributed hash tables is presented by Kaoudi et al. [KMK08]. They give algorithms for both forward and backward chaining and an analytical cost model for querying and storage. Oren et al. describe a distributed RDFS reasoning system on a peer network [OKA$^+$09]. Their entailment is asymptotically complete since only a subset of the generated triples are forwarded to other nodes. Duplicate elimination is performed using bloom filters. Urbani et al. describe a parallel implementation of the RDFS rule set using the MapReduce framework [UKOvH09]. They introduce a topological order of the RDFS rules such that there is no need for fix point iteration.

Rule-based parallel reasoners for the OWL 2 EL profile have been presented for TBox [KKS11] and ABox reasoning [RPL11]. Both works rely on a shared queue for centralised task management. [UKM$^+$10] presents an approach that allows computing the OWL closure over billion of triples. In [MNP$^+$14], an approach for parallel OWL 2 RL reasoning in centralised systems is demonstrated. If feasible, we would like to see more of those approaches supporting the OWL-API so that those reasoners could be tested with the DL-Learner framework.

Strongly related to our SPARQL bridge is [BKPR14], where the authors show that every RDF database featuring SPARQL 1.1 queries can be used as an OWL QL reasoner. We plan on integrating and comparing this rewriting method in the future. Both that work and our method are different from approaches such as Sparql-DL [SP07], which attempt to embed access to OWL-API via SPARQL query language extensions.

7 Conclusion

In this paper, we presented an OWL to SPARQL bridge, which allows structured machine learning to be carried out on large amounts of data. A theoretical result for the bridge is proven, which shows that the rewriting approach is plausible in the considered concept induction scenario. We showed how this bridge can be combined with the CELOE machine learning approach and evaluated this approach on four data sets. We are contributing a way of using supervised structured machine learning directly on semantic data endpoints accessible via SPARQL. The approach shows good scalability results. In particular, the SPARQL-based querying outperforms the reasoner-based implementation within the same runtime. Most importantly, it is able to work on a comparably large data set which fails to load into OWL-API-compatible reasoners such as Pellet and HermiT. This result is of capital importance as it suggests that our bridge makes learning on large OWL data sets possible, which has not been practically shown before. Reasoning via SPARQL endpoint lays the foundation to use big data sets stored in endpoints for machine learning. Hence, it seems to be a viable path towards the implementation of the Semantic Web vision. The results presented herein are part of larger research agenda, in which we aim to build up an RDF/OWL-based scalable structured machine learning pipeline for Web-scale data sets such as DBpedia, LinkedTCGA and LinkedGeoData.

Acknowledgements

This work was supported in part by a research grant from the German Ministry for Finances and Energy under the SAKE project (Grant No. 01MD15006E) and by a grants from the EU's 7th Framework Programme provided for the projects GeoKnow (GA no. 318159), Big Data Europe (GA no. 644564) and HOBBIT (GA no. 688227).

Bibliography

[AGP10] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. On the semantics of SPARQL. In *Semantic Web Information Management*, pages 281–307. Springer, 2010.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[BCM$^+$03a] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniel Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[BCM$^+$03b] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BKPR14] Stefan Bischof, Markus Krötzsch, Axel Polleres, and Sebastian Rudolph. Schema-agnostic query rewriting in SPARQL 1.1. In *The Semantic Web–ISWC 2014*, pages 584–600. Springer, 2014.

[BL13] Lorenz Bühmann and Jens Lehmann. Pattern based knowledge base enrichment. In *The Semantic Web – ISWC 2013*, volume 8218 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin Heidelberg, 2013.

[Bra78] Ronald J. Brachman. A structural paradigm for representing knowledge. Technical Report BBN Report 3605, Bolt, Beraneck and Newman, Inc., Cambridge, MA, 1978.

[Bra99] Ivan Bratko. *Inductive Logic Programming: 9th International Workshop, ILP-99 Bled, Slovenia, June 24–27, 1999 Proceedings*, chapter Refining Complete Hypotheses in ILP, pages 44–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[CDGL$^+$05] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *AAAI*, volume 5, pages 602–607, 2005.

[CH92] William W. Cohen and Haym Hirsh. Learnability of description logics. In *Proceedings of the 4th Annual Workshop on Computational Learning Theory*. ACM Press, 1992.

[DLdCD$^+$91] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[ESWW07] M Elter, R Schulz-Wendtland, and T Wittenberg. The prediction of breast cancer biopsy outcomes using two cad approaches that both emphasize an intelligible decision process. *Medical Physics*, 34(11):4164–4172, 2007.

[FdE08] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL: Concept learning in description logics. In F. Zelezný and N. Lavrac, editors, *Proc. of the 18th Int. Conf. on Inductive Logic Programming (ILP)*, volume 5194 of *LNAI*, pages 107–121. Springer, 2008.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 57–67. AAAI Press, 2006.

[HLA09] Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of OWL class descriptions on very large knowledge bases. *International Journal on Semantic Web and Information Systems*, 5(2):25–48, 2009.

[IPF07] Luigi Iannone, Ignazio Palmisano, and Nicola Fanizzi. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159, 2007.

[KKS11] Y Kazakov, M. Krötzsch, and F Simančík. Concurrent Classification of EL Ontologies. In *International Semantic Web Conference*, pages 305–320. Springer, 2011.

[KM94] Jörg Uwe Kietz and Katharina Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(2):193–218, 1994.

[KMK08] Z. Kaoudi, I. Miliaraki, and M. Koubarakis. RDFS Reasoning and Query Answering on Top of DHTs. In *International Semantic Web Conference*, pages 499–516. Springer, 2008.

[LABT11] Jens Lehmann, Sören Auer, Lorenz Bühmann, and Sebastian Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9:71 – 81, 2011.

[Leh07] Jens Lehmann. Hybrid learning of ontology classes. In *Proc. of the 5th Int. Conference on Machine Learning and Data Mining MLDM*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer, 2007.

[Leh09] Jens Lehmann. DL-Learner: learning concepts in description logics. *Journal of Machine Learning Research (JMLR)*, 10:2639–2642, 2009.

[LH09] Jens Lehmann and Christoph Haase. Ideal downward refinement in the EL description logic. In *Proc. of the Int. Conf. on Inductive Logic Programming*, volume 5989 of *LNCS*, pages 73–87. Springer, 2009.

[LH10] Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250, 2010.

[Mit82] T M Mitchell. Generalisation as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[MNP$^+$14] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel OWL 2 RL materialisation in centralised, main-memory RDF systems. In *Informal Proceedings of the 27th International Workshop on Description*, pages 311–323, 2014.

[NCdW97] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf, editors. *Foundations of Inductive Logic Programming*, volume 1228 of *LNCS*. Springer, 1997.

[OKA$^+$09] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. Ten Teije, and F. van Harmelen. Marvin: A platform for large-scale analysis of Semantic Web data. In *Proc. of the WebSci'09*, 2009.

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.

[Qui90] J. Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990.

[RPL11] Y. Ren, J.Z. Pan, and K. Lee. Parallel ABox Reasoning of EL Ontologies. In *Proc. of the First Joint International Conference of Semantic Technology (JIST 2011)*, 2011.

[SKI$^+$14] Muhammad Saleem, Maulik R Kamdar, Aftab Iqbal, Shanmukha Sampath, Helena F Deus, and Axel-Cyrille Ngonga Ngomo. Big linked cancer data: Integrating linked tcga and pubmed. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27:34–41, 2014.

[SP07] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *OWLED*, volume 258, 2007.

[SPG$^+$08] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Abstract Pellet: A practical OWL-DL reasoner, 2008.

[SPN$^+$13] Muhammad Saleem, Shanmukha Sampath Padmanabhuni, Axel-Cyrille Ngonga Ngomo, Jonas S. Almeida, Stefan Decker, and Helena F. Deus. Linked cancer genome atlas database. In *Proceedings of I-Semantics2013*, 2013.

[TH06] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[UKM$^+$10] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank Van Harmelen, and Henri Bal. OWL reasoning with WebPIE: calculating the closure of 100 billion triples. In *The Semantic Web: Research and Applications*, pages 213–227. Springer, 2010.

[UKOvH09] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Hermelen. Scalable Distributed Reasoning using MapReduce. In *The Semantic Web – ISWC 2009*, 2009.

[VN11] Johanna Völker and Mathias Niepert. Statistical schema induction. In *Proc. of the Extended Semantic Web Conference (ESWC)*, pages 124–138, 2011.

[WH09] J. Weaver and J. Hendler. Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. In *The Semantic Web – ISWC 2009*, pages 682–697. Springer, 2009.