Semantic Web 2 (2011) 71–87
DOI 10.3233/SW-2011-0034
IOS Press

Comparison of reasoners for large ontologies in the OWL 2 EL profile

Editor(s): Bernardo Cuenca Grau, Oxford University, UK
Solicited review(s): Julian Mendez, Dresden University of Technology, Germany; anonymous reviewer

Kathrin Dentler [a,b,*], Ronald Cornet [a], Annette ten Teije [b] and Nicolette de Keizer [a]
[a] *Department of Medical Informatics, Academic Medical Center, University of Amsterdam, The Netherlands*
[b] *Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands*

Abstract. This paper provides a survey to and a comparison of state-of-the-art Semantic Web reasoners that succeed in classifying large ontologies expressed in the tractable OWL 2 EL profile. Reasoners are characterized along several dimensions: The first dimension comprises underlying reasoning characteristics, such as the employed reasoning method and its correctness as well as the expressivity and worst-case computational complexity of its supported language and whether the reasoner supports incremental classification, rules, justifications for inconsistent concepts and ABox reasoning tasks. The second dimension is practical usability: whether the reasoner implements the OWL API and can be used via OWLlink, whether it is available as Protégé plugin, on which platforms it runs, whether its source is open or closed and which license it comes with. The last dimension contains performance indicators that can be evaluated empirically, such as classification, concept satisfiability, subsumption checking and consistency checking performance as well as required heap space and practical correctness, which is determined by comparing the computed concept hierarchies with each other. For the very large ontology SNOMED CT, which is released both in stated and inferred form, we test whether the computed concept hierarchies are correct by comparing them to the inferred form of the official distribution. The reasoners are categorized along the defined characteristics and benchmarked against well-known biomedical ontologies. The main conclusion from this study is that reasoners vary significantly with regard to all included characteristics, and therefore a critical assessment and evaluation of requirements is needed before selecting a reasoner for a real-life application.

Keywords: Semantic Web, DL reasoners, OWL 2 EL, biomedical ontologies, SNOMED CT

1. Introduction

Ontologies are formal definitions of concepts and the relationships between them. The ontology language OWL 2[1] is a W3C Recommendation since 2009. It is based on Description Logics (DLs) [7], a family of knowledge representation formalisms. OWL 2 has three tractable profiles[2], i.e. logical fragments that trade expressive power for the efficiency of reasoning. Each profile is restricted to a different sublanguage of OWL 2. Which profile to choose for a given application scenario depends on the structure of the employed ontology and on the required reasoning tasks. The three profiles are OWL 2 RL, OWL 2 QL and OWL 2 EL. OWL 2 RL (Rule Language) reasoning systems allow for rule-based reasoning. OWL 2 QL (Query Language) supports conjunctive query answering against large volumes of instance data that is stored in relational database systems. OWL 2 EL aims at applications that employ large ontologies. This profile is sufficiently expressive for many biomedical ontologies, such as the very large ontology SNOMED CT [13], and basic reasoning problems for OWL 2 EL can be decided in polynomial time. As indicated by its acronym EL, the profile is based on the $\mathcal{EL}$ family of

*Corresponding author. E-mail: k.dentler@vu.nl.
[1] http://www.w3.org/TR/owl2-overview/
[2] http://www.w3.org/TR/owl2-profiles/

description logics that provide only existential (and no universal) quantification.

A reasoner is a program that infers logical consequences from a set of explicitly asserted facts or axioms and typically provides automated support for reasoning tasks such as classification, debugging and querying. For OWL 2 EL, scalable implementations of dedicated reasoning algorithms are available. A question is whether these implementations perform better on OWL 2 EL ontologies than traditional reasoning engines, which have been designed for much more expressive languages. Tableau algorithms can be highly optimized [7], so that they are not necessarily outperformed by straightforward implementations of polynomial-time algorithms [3].

Ontologies consist of two different types of statements: TBox statements describe intensional knowledge, that is terminological background knowledge, while ABox statements describe extensional knowledge about individuals. The experiments of this study are limited to (very large) TBoxes.

The main contribution of this paper is the *identification of reasoner characteristics* that influence the choice of a particular reasoner for a given application scenario. A second contribution is the *categorization of dedicated OWL 2 EL and tableau-based reasoners along these characteristics*. To categorize the reasoners along performance indicators, a benchmark is employed that is based on three biomedical ontologies and comprises several TBox reasoning tasks. This categorization can be used to make a well-motivated choice for a particular application. The remainder of this paper is organized as follows: The next section summarizes related work. Section 3 gives an overview of characteristics that are relevant to compare reasoning engines. Those characteristics are grouped in the three dimensions reasoning characteristics, practical usability and performance indicators. Section 4 presents the eight reasoners that are included in this study. Section 5 contains the classification of the reasoners as well as the experimental results on their performance. Section 6 discusses the results and their implications.

2. Related work

A benchmark typically comprises a selection of employed ontologies and a number of standard TBox and ABox reasoning tasks and serves as a basis to evaluate and compare reasoners. With the increasing availability of reasoners for OWL and OWL 2 EL, several benchmarks have been proposed.

The Lehigh University Benchmark (LUBM) [17] and the University Ontology Benchmark (UOBM) [34], which is an extension of the LUBM, are based on synthetically generated ontologies. LUBM evaluates the performance of answering conjunctive queries over an ABox of varying size that commits to an OWL Lite ontology. Additionally, LUBM measures correctness by examining query completeness and soundness.

A framework for an automated comparison of DL reasoners that focusses on TBox classification is presented in [15]. It is based on real-life ontologies and allows users to compare the classification performance of reasoners as well as to analyze the "correctness" of classification by comparing computed concept hierarchies. This benchmarking system is based on the DIG standard [53], which facilitates the comparison of DIG-compliant reasoners such as FaCT++ [52], KAON2 [38], Pellet [45] and RacerPro [19].

The authors of [9] aim at providing guidance for the nontrivial task of choosing an appropriate reasoner for a given application scenario. The paper surveys the ontology landscape and defines a benchmark, which includes classification as representative TBox reasoning task and conjunctive query answering as ABox reasoning task. Employed performance measures contain load time and response time for ontologies that are representative for identified language fragments. The OWL 2 EL fragment is not included in this study, but the authors state that the investigation of tractable fragments of OWL and the development of reasoners specialized for these fragments is an important research topic. Reasoners are grouped into three categories according to their underlying reasoning techniques: tableau-based algorithms (HermiT [43], RacerPro and Pellet), datalog engines (KAON2) and standard rule engines (Sesame [11] and OWLIM [26]).

A comprehensive survey of OWL reasoners that aims to serve as a decision help for Semantic Web application designers is provided by [31]. Reasoners are described in the categories "official OWL specification language conformity", correctness, efficiency, interface capabilities and inference services. Included reasoners are FaCT++, RacerPro, Pellet, KAON2 and Hoolet, an OWL DL reasoner that uses the first-order theorem prover Vampire [41]. The correctness of reasoners is evaluated by running inference test cases for selected language features.

The survey [33] employs a benchmark suite for large ABox data, as well as a selection of small but difficult

T- and ABox test cases. It analyzes the correctness of the results of FaCT++, Pellet, RacerPro, KAON2 and HermiT. The authors extend the challenge of finding an optimal OWL reasoner for a specific application by finding an optimal service interface. The performance of several protocols is compared in different computing environments, which leads to the conclusion that those components may have a high impact. Additionally, the paper contains a feature matrix of selected system characteristics including available interfaces such as the OWL API, Jena [12], DIG and OWLlink, language support in terms of expressivity, retraction, incremental reasoning, SWRL support, query language and query entailment, as well as available licenses and implementation language.

The SEALS (Semantic Evaluation at Large Scale) project provides an infrastructure to evaluate semantic technologies. Its Storage and Reasoning Systems Evaluation Campaign 2010[3] includes evaluation scenarios for standard inference services such as classification, concept satisfiability, ontology satisfiability and logical entailment. In the scope of the 2010 campaign, the reasoning engines HermiT, FaCT++ and jcel[4] have been evaluated based on an OWL 2 repository and widely-used real-world ontologies.

Recently, Mishra et al. [36] presented an extensive survey of nineteen reasoners that have been released between 1975 and 2009. The authors compare these reasoners with respect to their inference support, completeness and algorithm, implementation language and supported Semantic Web languages.

Developers of dedicated OWL 2 EL reasoners have been comparing their classification performance to other reasoners. All these comparisons employ life-science ontologies in OWL 2 EL as benchmark ontologies: the Gene Ontology (GO), a large ontology from the US National Cancer Institute (NCI), the Foundational Model of Anatomy (FMA), the Generalized Architecture for Languages, Encyclopaedias and Nomenclatures in medicine (GALEN) and the Systematized Nomenclature of Medicine, Clinical Terms (SNOMED CT). All mentioned experiments except [47] (that summarizes common characteristics of several life-science ontologies and suggests that the use of DLs in the $\mathcal{EL}$ family is beneficial both in terms of expressivity and of scalability, and also promotes CEL's reasoning services) have been performed with the goal to demonstrate that the respective newly introduced or re-introduced reasoner outperforms existing reasoners, with TBox classification performance as the only dimension for comparison. In the following, the classification performance with regard to SNOMED CT measured in these studies is briefly outlined.

CEL (Classifier for EL) [35] has been compared to FaCT and Racer in 2005 [3]. In this study, CEL was the only reasoner successful in classifying SNOMED CT, which took around 3.5 hours. FaCT and Racer failed due to memory exhaustion (the test machine had 2GB memory). The fact that CEL succeeded in classifying SNOMED CT motivated the DL community to investigate optimizations that exploit the simple structure of biomedical ontologies. In 2006, CEL was compared to FaCT++, RacerMaster and Pellet [5]. CEL and FaCT++ were successful in classifying SNOMED CT, while RacerMaster and Pellet failed (512MB; Java heap space set to 256MB). FaCT++ needed a little more than an hour and CEL completed just under half an hour. CEL was compared to FaCT++, HermiT, KAON2, Pellet and RacerPro in 2008 [47]. Here, CEL (around 20 minutes), FaCT++ (around 10 minutes) and RacerPro (around 20 minutes) succeeded. KAON2 failed due to a timeout after 24 hours, while HermiT and Pellet failed due to memory exhaustion (machine with 2GB memory, heap space set to 1.5GB). The same results are presented in [35]. The consequence-based reasoner CB has been compared to FaCT++, Pellet, HermiT and CEL in 2009 [25]. CB classified SNOMED CT in less than a minute, FaCT++ in around 10 minutes and CEL in around 20 minutes, while HermiT and Pellet failed to return a result (1.5GB RAM, 1GB heap space; timeout 1 hour). Finally, various Protégé Plugins (Snorocket, FaCT++, Pellet and CEL) have been compared in 2010 [30]. CEL and Pellet failed due to memory exhaustion (4GB memory, 1,900MB maximum heap space). Snorocket classified SNOMED CT in under a minute and FaCT++ in around 20 minutes. Figure 1 shows how the classification performance for SNOMED CT has been improving over the recent years. Only successful outcomes (i.e. no timeout or memory exhaustion results) are included in this figure.

Classification performance is indeed essential and the fact that very large ontologies such as GALEN or SNOMED CT can be classified at all and within a reasonable time is a remarkable achievement of the recent years. But when a reasoner is to be applied in a real-world setting, many more orthogonal aspects are rel-

[3] http://www.seals-project.eu/seals-evaluation-campaigns/storage-and-reasoning

[4] http://jcel.sourceforge.net/

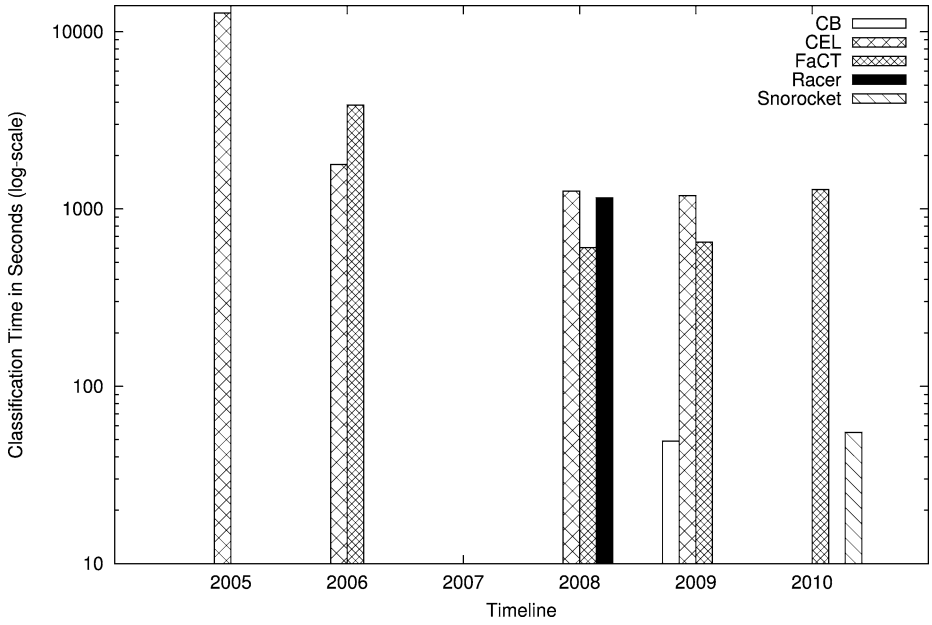

Fig. 1. Classification performance for SNOMED CT over time.

evant. In the following section, we will identify those characteristics and group them into three dimensions.

3. Characteristics

The characteristics described in this section stem from a literature review that included the related work. We analyzed papers that describe the reasoners contained in this study (see Section 4) as well as short advertising descriptions of reasoners, which usually outline the respective reasoner's strong points. Additionally, many characteristics in the dimension of practical usability arose while the reasoning experiments have been performed. The characteristics are arranged in three dimensions: reasoning characteristics, practical usability and performance indicators.

3.1. Dimension reasoning characteristics

Methodology Most DL reasoners are based on (hyper)tableau calculi [23,37], which are sound and complete. Such procedures aim at large expressivity and, according to [25], classify an ontology by iterating over all necessary pairs of concepts and trying to build a model of the ontology that violates the subsumption relation between them. New kinds of reasoning procedures have been developed for less expressive, tractable DLs such as $\mathcal{EL}^{++}$ [1,2]. The procedure for $\mathcal{EL}^{++}$ infers subsumption relations by using so-called completion rules in a goal-directed way.

Soundness and completeness in theory This property evaluates whether the inferences of the employed reasoning methods are sound based on the underlying semantics and whether they are complete, i.e. whether all possible inferences are inferred. Soundness or completeness can be sacrificed for a significant speed-up of reasoning [42]. Thus, to employ a reasoner in a real-world application, it is not always important *that* its underlying reasoning method is sound and complete, but it is important to know *whether* it is sound and complete. Most of the methods underlying the reasoners included in this study have been proven to be sound and complete, i.e. correct. This does not imply that their implementations are correct.

Expressivity and computational complexity For description logics, a tradeoff exists between logical expressivity and computational complexity: the more expressive a language, the higher its computational complexity. Reasoning problems in OWL DL and OWL 2 are, in the worst case, solvable in time that is (double) exponential with respect to the size of the input. However, hard cases that lead to worst-case behavior rarely occur in practice [22,23]. When the input ontology is in a tractable profile such as OWL 2 EL, it is theoretically possible for reasoners that support a more expressive language to terminate in polynomial time. Table 1

Table 1
Worst-case complexities of concept satisfiability checking

Logic	Worst-Case Complexity
$\mathcal{EL}^{++}$	PTime [1,2]
Horn $\mathcal{SHIQ}$	ExpTime[25]
$\mathcal{SHIQ}$	ExpTime [51]
$\mathcal{SHOIQ}$ (OWL DL)	NExpTime [51]
$\mathcal{SROIQ}$ (OWL 2)	N2ExpTime [24]

lists several DLs with their corresponding worst-case complexities for concept satisfiability checking taken from the literature.

The expressivity of a particular DL is determined by the concept constructors it provides. The informal naming convention for DLs describes the constructors that can be used: $\mathcal{E}$ (existential restrictions), $\mathcal{Q}$ (qualified number restrictions), $\mathcal{O}$ (nominals, objects), $\mathcal{I}$ (inverse roles), $\mathcal{H}$ (role hierarchies) and $\mathcal{S}$ is the abbreviation for $\mathcal{ALC}$ with transitive roles. The basic description logic $\mathcal{ALC}$ uses the constructors $\neg C$ (negation), $C \sqcap D$ (conjunction), $C \sqcup D$ (disjunction), $\exists R.C$ (existential restriction) and $\forall R.C$ (value restriction).

The description logic $\mathcal{EL}^{++}$ The language $\mathcal{EL}$ [10] allows for concepts constructed from atomic concepts A and the top concept $\top$ (i.e. owl:Thing) by using the constructors conjunction $C \sqcap D$ (i.e. owl:ObjectIntersectionOf) and existential restriction $\exists r.C$ (i.e. owl:ObjectSomeValuesFrom), where r is an atomic role. Axioms of $\mathcal{EL}$ are general concept inclusions (GCI) $C \sqsubseteq D$. A primitive concept definition (PCD, necessary but not sufficient) $A \sqsubseteq D$ is a GCI with a concept name on the left-hand side, while a full concept definition (FCD, necessary and sufficient) $A \equiv D$ can be expressed by the two GCIs $A \sqsubseteq D$ and $D \sqsubseteq A$. A finite set of GCIs is called a TBox. Concepts and roles correspond to OWL classes and properties. $A \sqsubseteq D$ corresponds to owl:SubClassOf(A, D) and $A \equiv D$ corresponds to owl:EquivalentClasses(A, D).

$\mathcal{EL}^{+}$ [3] extends $\mathcal{EL}$ by complex role inclusions which allow to express role hierarchies, transitive roles and right identities, such as: $r \circ s \sqsubseteq t$, e.g. has-parent $\circ$ has-sister $\sqsubseteq$ has-aunt. The example expresses that two individuals that are connected by a chain of roles (has-parent $\circ$ has-sister) are necessarily connected by the role on the right-hand side (has-aunt). $\mathcal{EL}^{+}$ is sufficiently expressive for many well-known biomedical ontologies such as the ones in our test-suite.

$\mathcal{EL}^{++}$ [1,2] extends $\mathcal{EL}^{+}$ by nominals (and thus ABoxes), the bottom concept $\bot$ (and thus disjointness constraints on concepts in the form of $C \sqcap D \sqsubseteq \bot$), reflexive roles and range restrictions $range(r) \sqsubseteq C$ (a global syntactic restriction applies to guarantee polynomiality) and a restricted form of concrete domains (e.g. references to numbers and strings; datatypes in OWL). $\mathcal{EL}^{++}$ is one of the few description logics for which standard reasoning problems such as ontology consistency, concept subsumption, and instance checking are decidable in polynomial time. To gain this tractability, commonly-used constructors such as universal value restrictions and inverse and functional roles have been sacrificed. For most biomedical ontologies, scalable reasoning seems to be more important than the expressivity of the language [49]. A complete description of $\mathcal{EL}^{++}$ and its formal semantics is given in [2], and the structure of OWL 2 EL ontologies is specified in the OWL 2 EL profile specification[5]. Mappings between $\mathcal{EL}^{++}$ and OWL 2 EL can be found in [35]. Further tractable extensions on $\mathcal{EL}^{++}$ exist [28,29].

Fracture $\sqsubseteq$ *Traumatic abnormality by morphology*
Traumatic abnormality by morphology $\sqsubseteq$ *Traumatic abnormality*
Traumatic abnormality $\sqsubseteq$ *Damage*

Bone structure of foot $\sqsubseteq$ *Bone structure of ankle and/or foot*
Bone structure of ankle and/or foot $\sqsubseteq$ *Bone structure of lower limb*

Fracture of bone $\equiv$ *Disorder of bone* $\sqcap \exists$*rolegroup.*
($\exists$*associated morphology.Fracture*
$\sqcap \exists$*finding site.Bone structure)*

Fracture of lower limb $\equiv$ *Fracture of bone* $\sqcap \exists$*rolegroup.*
($\exists$*associated morphology.Fracture*
$\sqcap \exists$*finding site.Bone structure of lower limb)*

Fracture of foot $\equiv$ *Fracture of lower limb* $\sqcap \exists$*.rolegroup*
($\exists$*associated morphology.Fracture*
$\sqcap \exists$*finding site.Bone structure of foot)*

Inferred:

Fracture $\sqsubseteq$ *Damage*
Bone structure of foot $\sqsubseteq$ *Bone structure of lower limb*
Fracture of foot $\sqsubseteq$ *Fracture of lower limb*

Pellet's explanation for *Fracture of foot* $\sqsubseteq$ *Fracture of lower limb*:

Fracture of foot $\equiv$ *Fracture of lower limb* $\sqcap \exists$*.rolegroup*
($\exists$*associated morphology.Fracture*
$\sqcap \exists$*finding site.Bone structure of foot)*

Fig. 2. An example $\mathcal{EL}$ ontology (motivated by SNOMED CT).

As an example, let us consider the ontology in Fig. 2. In SNOMED CT, concepts form logical groupings, which are expressed by nested existential restrictions. All concepts that are allowed to be grouped are included under an existential restriction that represents the (potential) grouping. This restriction is labeled with an owl:ObjectProperty named rolegroup.

[5] http://www.w3.org/TR/owl2-profiles/#OWL_2_EL

Attributes (i.e. Object Properties or roles) are presented in lower case. The figure contains five partially defined concepts (Fracture, Traumatic abnormality by morphology and Traumatic abnormality, Bone structure of foot and Bone structure of ankle and/or foot) that form hierarchies. The three fully defined concepts are Fracture of bone, Fracture of lower limb and Fracture of foot. It can be inferred that a Fracture is a Damage, i.e. *Fracture* $\sqsubseteq$ *Damage*, that *Bone structure of foot* $\sqsubseteq$ *Bone structure of lower limb* and that Fracture of foot is a Fracture of lower limb, i.e. *Fracture of foot* $\sqsubseteq$ *Fracture of lower limb*.

Incremental classification When an ontology has been classified and is updated afterwards (by additions or removals), it makes sense for a reasoner to reuse the previous classification information together with the updated axioms to produce the new concept hierarchy. This is especially reasonable in typical ontology development scenarios that involve only minor modifications between classifications that are performed to check whether the developed ontology is (still) consistent. Alternatively, the reasoner has to re-start the whole classification from scratch, which can be time-consuming.

Rule support Rule support enables the combination of ontologies with rules. Some reasoners support SWRL[6] rules. SWRL, the Semantic Web Rule Language, extends the set of OWL axioms to include Horn-like rules. A simple exemplary rule is to assert that the combination of the *hasParent* and *hasBrother* properties implies the *hasUncle* property: *hasParent(?x1,?x2)* $\wedge$ *hasBrother(?x2,?x3)* $\Rightarrow$ *hasUncle(?x1,?x3)*. In contrast to most other rules, this simple example can also be expressed by a complex role inclusion (i.e. owl:ObjectPropertyChain).

Justifications Justifications are minimal entailing subsets of an ontology [21]. Given an ontology and an unclear consequence, may it be a subsumption relationship or an unsatisfiable concept, it can be very helpful if a reasoner computes a justification (or all justifications) for the consequence, which can subsequently be used to explain or debug that consequence. The OWL API contains a method that returns all explanations for a given unsatisfiable concept, or an empty set if the concept is satisfiable.

[6] http://www.w3.org/Submission/SWRL/

Support of ABox reasoning tasks ABox reasoning is reasoning with individuals and comprises instance checking, (conjunctive) query answering and ABox consistency checking. Instance checking tests whether a knowledge base entails that an individual is an instance of a concept. It is the basis of query answering, which can be performed by iterating instance checking for all individuals in a knowledge base [14]. Whether a reasoner supports ABox reasoning tasks or not is a characteristic that, depending on the intended application, can be very relevant.

3.2. Dimension practical usability

OWL API The OWL API [20] is an Application Programming Interface (API) for working with OWL ontologies. It supports parsing and writing in the syntaxes that are defined in the OWL 2 specification. The open source reference implementation in Java includes validators for OWL 2 profiles. It also provides a standard interface to OWL reasoners, so that an application can embed different reasoners without having to change its implementation. A number of existing reasoners provide OWL API wrappers and are thus easily integrated into OWL API based applications such as Protégé 4.

OWLlink OWLlink [32] provides an extensible, implementation-neutral protocol to interact with OWL 2 reasoners. It succeeds the DL-oriented DIG interface. OWLlink facilitates client applications to manage reasoners, to assert axioms and to access reasoning services via a set of standard queries. The OWL API based OWLlink API implements the OWLlink protocol. It allows to turn OWL API aware reasoners into OWLlink servers and to access remote OWLlink servers from OWL API based applications (such as Protégé).

Availability as Protégé plugin Protégé is an open source ontology editor. The new version 4.1 fully conforms with the OWL 2 language specification and is built on top of the OWL API. It is a common practice of reasoner developers to release a plugin for Protégé. OWL API aware reasoners can also be used from Protégé via OWLlink.

License Many reasoners come with a dual license. This means that they are free under certain conditions, and that for different use, arrangements have to be made with their developers. The major distinguishing

feature concerning licenses is whether the license is a recognized open source license[7] or not.

Further characteristics The remaining characteristics are self-explanatory and include whether the source of the reasoner is open or closed, the programming language the reasoner is implemented in, the supported platforms, whether the reasoner has a native Jena[8] interface and the kind of institution (academic, governmental or commercial) it has been developed in. Jena is a Java framework for building Semantic Web applications.

3.3. Dimension performance indicators

The dimension performance indicators contains characteristics that depend on the input ontology and that can be measured empirically. At the ontology level, fundamental reasoning services include classification and consistency checking. The two most important reasoning services at the concept level are satisfiability checking and subsumption. In our experiments, performance indicators are measured based on these reasoning tasks. The performance of ABox reasoning tasks is not included in this study. Another characteristic that is included is the minimum required amount of heap space for Java reasoners. Finally, we analyze classification results in order to check whether the reasoners' theoretical correctness is confirmed in practice.

Classification performance Classification, i.e. the computation of the concept hierarchy, is one of the most important reasoning services and supported by all modern DL systems. Thus, its duration is often used as a performance indicator to benchmark reasoning engines. From a practical perspective, an ontology should be classified regularly during its development and maintenance in order to detect unwanted subsumptions as soon as possible. To make this feasible also for large ontologies, classification should be fast.

TBox consistency checking performance An interpretation I is a model of an ontology O if the interpretation satisfies all implications in O. An ontology is consistent if it has a model [4].

Concept satisfiability checking performance A concept satisfiability check tests whether a concept C can have instances. According to [7], satisfiability is formally defined as follows: A concept C is satisfiable with respect to a TBox T if there exists a model I of T such that C^{I} is nonempty. Concept satisfiability checks are a special case of concept subsumption checks, because a concept C is unsatisfiable if, and only if, $C \sqsubseteq \bot$ [49].

Subsumption query performance Subsumption queries check whether one concept subsumes another concept or return all concepts subsuming or subsumed by a concept. According to [7], a concept C is subsumed by a concept D with respect to T if $C^{\mathrm{I}} \subseteq D^{\mathrm{I}}$ for every model I of T. This is written as $T \vDash C \sqsubseteq D$.

Soundness and completeness in practice We analyze the output of reasoners in Section 5. For most ontologies, their closure, i.e. the set of all statements that follow from the underlying semantics, is not given. In such cases, the only way to evaluate the output of a reasoner is by comparing it to the output of other reasoners. All reasoners that correctly implement a sound and complete reasoning method that supports the expressivity of the input ontology should produce the same output for the same input. Thus, if the outputs of the reasoners differ from each other, we can infer that not all implemented methods are sound and complete in practice. For SNOMED CT, an advantageous situation applies: It is released both in stated and in inferred form, so that we can employ the inferred form as gold standard and compare it to the concept hierarchies computed by the reasoners. The inferred form has been generated with Apelon's[9] Ontylog DL classifier.

4. Reasoners

The reasoners which are compared based on the defined characteristics include the newly introduced reasoner TrOWL and all reasoners that occur in previous comparisons except KAON2, because it is not being maintained any longer. This section briefly describes each reasoner.

CB (Consequence-based reasoner, University of Oxford) is an implementation of a reasoning procedure [25] for Horn $\mathcal{SHIQ}$ ontologies, i.e. $\mathcal{SHIQ}$ ontologies that can be translated to the Horn fragment of first-order logic. CB's reasoning procedure can be regarded as an extension of the completion-based procedure for $\mathcal{EL}^{++}$ ontologies and works by deriving new consequent axioms. It is theoretically optimal for Horn

[7] http://www.opensource.org/licenses
[8] http://jena.sourceforge.net/
[9] http://www.apelon.com

$\mathcal{SHIQ}$ ontologies as well as for the common fragment of $\mathcal{EL}^{++}$ and $\mathcal{SHIQ}$ [25].

CEL (Classifier for $\mathcal{EL}$, TU Dresden) [5,35] implements a refined polynomial-time algorithm [1–3] which allows it to process very large $\mathcal{EL}^{+}$ ontologies in reasonable time.

FaCT++ (Fast Classification of Terminologies, University of Manchester) [52] is the new generation of the OWL DL reasoner FaCT. It supports OWL DL and a subset of OWL 2 that is more expressive than the ontologies in our test suite. FaCT++ is implemented in C++ and based on optimized tableaux algorithms.

HermiT (University of Oxford) [43] can determine whether or not a given ontology is consistent and identify subsumption relationships between concepts, among other features. HermiT is based on a "hypertableau" calculus.

Pellet (Clark & Parsia) [45] was the first reasoner that supported all of OWL DL ($\mathcal{SHOIN}(\mathcal{D})$) and has been extended to OWL 2 ($\mathcal{SROIQ}(\mathcal{D})$). Pellet supports OWL 2 profiles including OWL 2 EL.

RacerPro (Renamed ABox and Concept Expression Reasoner, Racer Systems) [19] implements the description logic $\mathcal{SHIQ}$. Dedicated optimizations for OWL 2 EL have been added (structural subsumption tests [18]), enabling practical reasoning with SNOMED CT.

Snorocket (CSIRO) [30] is a high-performance implementation of the polynomial-time classification algorithm for $\mathcal{EL}^{+}$ [3]. It was primarily optimized for classifying SNOMED CT, and was licensed to the IHTSDO[10] for integration into the Workbench software used to maintain and produce SNOMED CT.

TrOWL (Tractable reasoning infrastructure for OWL 2, University of Aberdeen) [50] is the common interface to a number of reasoners. TrOWL Quill provides reasoning services over OWL 2 QL. TrOWL REL is an optimized implementation of the CEL algorithm that provides reasoning over OWL 2 EL. It employs a syntactic approximation from OWL 2 DL to OWL 2 EL to enable OWL 2 DL ontologies to be classified within polynomial time [40]. This approximation is soundness-preserving but sacrifices completeness. To support full DL reasoning, TrOWL allows for the use of heavyweight plugin reasoners, such as FaCT++, Pellet, HermiT and RacerPro.

[10] http://www.ihtsdo.org/

5. Categorization of reasoners

In this section, the eight reasoners are categorized along the defined characteristics. The evaluation of the performance indicators is based on our test suite, which comprises the three biomedical ontologies GO, NCI and SNOMED CT. Concluding, we analyze the tradeoff between a reasoner's supported expressivity and its classification performance.

5.1. Dimension reasoning characteristics

Table 2 summarizes the reasoning properties for the included reasoners.

Methodology The first characteristic is the underlying reasoning methodology. Most reasoners rely on tableau-based methods or on (extensions of) completion rules for $\mathcal{EL}$. Pellet and TrOWL both implement optimized support for OWL 2 EL and their EL reasoners are activated based on the profile of the current ontology.

Soundness and completeness in theory Most of the underlying reasoning methodologies have been proven to be sound and complete. The tableaux and hypertableaux calculi are sound and complete, the procedure for $\mathcal{EL}^{++}$ has been shown to be sound and complete in [1] and the procedure for Horn $\mathcal{SHIQ}$ is sound and complete according to [25]. TrOWL REL is based on the procedure for $\mathcal{EL}^{++}$. TrOWL REL's syntactic approximation from OWL 2 DL to OWL 2 EL is soundness-preserving but possibly incomplete.

Expressivity and computational complexity Table 2 lists the languages that the reasoners support. CB's reasoning procedure described in [25] supports Horn $\mathcal{SHIQ}$, but its implementation supports only Horn $\mathcal{SHIF}$, which is a subset of Horn $\mathcal{SHIQ}$ that does not include cardinality restrictions. The expressivity of TrOWL depends on its configuration. TrOWL REL implements $\mathcal{EL}^{++}$ without datatypes and supports $\mathcal{SROIQ}$ by approximation. If a third-party reasoner is used, then its supported expressivity is the one of this reasoner.

Incremental classification CEL, Pellet and Snorocket support incremental reasoning. CEL and Snorocket both have a partial incremental classification functionality that only supports additions. Pellet supports incremental classification and incremental consistency checking. Pellet's incremental classification is based on module extraction: The first time an ontology is

Table 2
Reasoning characteristics

	CB	CEL	FaCT++	HermiT	Pellet	RP	SR	TrOWL (REL)
Methodology	consequence-based	completion rules	tableau-based	hypertableau	tableau-based	tableau-based	completion rules	approximation (completion rules)
Soundness	+	+	+	+	+	+	+	+ (+)
Completeness	+	+	+	+	+	+	+	− (+)
Expressivity	Horn $\mathcal{SHIF}$	$\mathcal{EL}^+$	$\mathcal{SROIQ}(\mathcal{D})$	$\mathcal{SROIQ}(\mathcal{D})$	$\mathcal{SROIQ}(\mathcal{D})$	$\mathcal{SHIQ}(\mathcal{D}-)$	$\mathcal{EL}^+$	third-party reasoner (approximating SROIQ; subset of $\mathcal{EL}^{++}$)
Incremental Classification (addition/removal)	−/−	+/−	−/−	−/−	+/+	−/−	+/−	−/−
Rule Support	−	−	−	+ (SWRL)	+ (SWRL)	+ (SWRL, nRQL)	−	−
Justifications	−	+	−	−	+	+	−	−
ABox Reasoning	−	+	+	+	+ (SPARQL)	+ (SPARQL, nRQL)	−	+ (SPARQL)

RacerPro (RP) and Snorocket (SR) had to be abbreviated due to space limitations. + stands for yes and − for no.

classified, Pellet computes modules for each concept. A module is a subset of an ontology which captures "everything" an ontology has to say about a particular sub-signature of the ontology [8]. By current methods, modules contain all justifications for all entailments expressible in their signature [8]. When the concept hierarchy of the ontology is changed, Pellet reclassifies only the affected module. Pellet's incremental reasoning supports axiom addition and removal [39].

The reasoner interfaces of the OWL API facilitate reasoners to expose incremental reasoning support. The API allows a reasoner to listen for ontology changes and to either immediately processes them or to queue them to processes them later [20].

Rule support Only HermiT, Pellet and RacerPro offer rule support. All of them support SWRL. HermiT and Pellet support SWRL in the DL-Safe Rules notion, which means rules will be applied only to named individuals in the ontology. RacerPro partially supports SWRL. SWRL is mapped to nRQL, which is RacerPro's native rule and query language.

Justifications CEL, Pellet and RacerPro support justifications for inconsistent concepts. RacerPro allows to check an ontology for inconsistent (unsatisfiable) concepts and generates an explanation for each inconsistency. Pellet can give a justification for any inference which it can compute.

Support of ABox reasoning tasks In contrast to all other reasoners, CB and Snorocket do not support ABox reasoning tasks. The reasoners that implement OWL API reasoner interfaces should (in theory) support all ABox reasoning tasks that are specified by the OWL API, such as retrieving the set of individuals that have been asserted to be an instance of a concept or retrieving the asserted types of an individual. RacerPro supports nRQL ABox queries and Pellet, RacerPro and TrOWL support SPARQL queries. SPARQL[11], the Query Language for RDF, is a W3C Recommendation since 2008. SPARQL allows to query required and optional graph patterns along with their conjunctions and disjunctions, to test values, to constrain queries by source RDF graph and to specify whether the result should be an RDF graph or a set.

5.2. Dimension practical usability

Table 3 shows how the reasoners are categorized along the defined usability characteristics.

OWL API All reasoners except CB are accessible via the OWL API, which is advantageous for applications that wish to access several reasoners via the same interface. The use of the OWL API highly facilitated the execution of our experiments.

[11] http://www.w3.org/TR/rdf-sparql-query/

Table 3
Practical usability

	CB	CEL	FaCT++	HermiT	Pellet	RP	SR	TrOWL
OWL API	−	+	+	+	+	+	+	+
OWLlink API	−	+	+	+	+	+	−	−
Protégé Plugin	−	+	+	+	+	−	+	+
License	DuLi: GLGPL	AP 2.0	GLGPL	GLGPL	DuLi: AGPL	own	own	DuLi: AGPL
Open Source	+	+	+	+	+	−	−	−
Language	OCaml	Common Lisp	C++	Java	Java	Lisp	Java	Java
Platforms	all	Linux	all	all	all	all	all	all
Jena	−	−	−	−	+	−	−	−
Institution	a	a	a	a	c	c	g	a

DuLi stands for dual license. + stands for yes and − for no. All platforms means that the reasoner is available for Windows, Linux, and Mac OS X. n/a abbreviates not applicable. Regarding the institution, a stands for academic, c for commercial and g for governmental.

OWLlink Most reasoners are accessible via OWLlink, and future protocol bindings might ease the integration of further reasoners like CB.

Protégé plugin All reasoners except CB and RacerPro can be plugged into Protégé. The RacerPro engine can be used as back-end inference system for Protégé via the RACER Protégé Plugin[12] or via OWLlink.

License CB can be redistributed and / or modified under the terms of the GNU Lesser General Public License (GLGPL) for non-commercial use. Pellet also comes with a dual license: software that is released under a recognized open source license can use Pellet under the terms of the Affero General Public License (AGPL), for other software, another license has to be arranged. This has the advantage that the community benefits from source code that uses Pellet under its open source license. TrOWL may be used under the terms of the AGPL for open source applications and is available under alternative license terms for proprietary, closed-source applications and other commercial applications. CEL comes with the Apache License 2.0 (AP 2.0), FaCT++ and HermiT with the GLGPL. Racer Systems offers several license types, including time-limited educational licenses, trial licenses and commercial licenses. Snorocket formulates its own license[13]. GLGPL, AGPL and AP 2.0 are open source licenses.

Further characteristics The remaining rows of Table 3 show further characteristics including whether the source of the reasoner is open or not and the programming language the reasoner is implemented in. Only Pellet has a native Jena interface. Further characteristics are the platforms the reasoner supports and the kind of institution (academic, governmental or commercial) it has been developed in.

5.3. Dimension performance indicators

In this section, the biomedical ontology test suite and the experimental setup are being presented. Subsequently, we present the results of our experiments.

Biomedical ontology test suite The biomedical ontologies presented in this section are well-established and have been used in previous benchmarks. All ontologies are in the tractable OWL 2 profile EL, so that especially in the case of SNOMED CT, the challenge for the reasoners lies in the sheer size of the ontologies. Consult [47] for additional information. The ontologies mainly differ in size, but also in whether they employ fully defined concepts or not. Biomedical ontologies are a typical use-case for OWL 2 EL, but this profile is also applicable in other domains where fast reasoning outweighs expressivity.

GO The Gene Ontology[14] project is an initiative which aims to standardize the representation of genes and gene product attributes. The Gene Ontology (GO) is a controlled vocabulary to describe gene product characteristics and annotation data.

NCI The National Cancer Institute thesaurus[15] is a terminology that covers clinical care and research, as well as public information and administrative activities.

[12] http://www.uni-ulm.de/in/ki/semantics/owltools
[13] http://research.ict.csiro.au/software/snorocket/LICENCE.txt
[14] http://www.geneontology.org/
[15] http://ncit.nci.nih.gov/

Table 4
Benchmark ontologies

	$\|N_{LA}\|$	$\|N_R\|$	$\|N_C\|$	PDC	FDC
GO	28,897	1	20,465	19,465	0
NCI	46,940	70	27,652	27,635	0
SNOMED CT	292,023	62	292,012	227,315	64,696

$|N_{LA}|$ is the number of logical axioms, $|N_R|$ the number of roles and $|N_C|$ the number of concepts. PDC stands for the number of primitively defined concepts and FDC for the number of fully defined concepts.

SNOMED CT SNOMED CT[16] consists of around 300,000 primitively and fully defined concepts. It is mainly used to represent clinical information in electronic health records. SNOMED CT contains one property chain (complex role inclusion) which is not used in the TBox.

GO, NCI and SNOMED CT can be regarded as acyclic $\mathcal{EL}$ TBoxes, i.e. sets of concept definitions without cyclic dependencies. GO has one transitive role, which is a special case of a role inclusion [1]. Also SNOMED CT is extended with role inclusion axioms. Table 4 provides an overview of the properties of the benchmark ontologies. The 1,000 concepts of GO that are neither fully nor primitively defined are just declared as concepts and thus direct subclasses of the top concept, without further definitions. 997 of those concepts are annotated as being obsolete and the 3 remaining concepts are the top-level concepts biological process, molecular function and cellular component. The 17 concepts in the NCI ontology that are neither fully nor primitively defined are Kinds, i.e. the top-level superclasses for all of the concepts defined in the thesaurus. They represent the possible categories that concepts can belong to, such as Anatomy, Biological Processes, Chemicals and Drugs, and Diagnostic and Prognostic Factors. In SNOMED CT, the root concept is neither fully nor partially defined.

Experimental setup For the experiments to measure performance indicators, the latest available versions of the included reasoners have been used: CB[17] build 6, CEL[18] plugin 0.4.0 for Protégé 4.1, FaCT++[19] 1.5.0, HermiT[20] 1.3.0, Pellet[21] 2.2.2, RacerPro[22] 2.0 pre-view, Snorocket Protégé plugin [23] version 1.3.2 and TrOWL[24] 0.5.1. We generated the SNOMED CT ontology with the OWL transformation script from the Stated Relationships Table of the latest (July 2010) SNOMED CT distribution. GO and NCI have been employed in other benchmarks and are available on http://reasonerben.ch.

For an ideal comparison, it would be desirable to run all reasoners via the same interface, such as the OWL API or OWLlink. Unfortunately, there is no interface that has been implemented by all reasoners. Thus, reasoners were tested separately: CB and RacerPro in batch mode and all other reasoners via the OWL API. For the reasoners which are called via the command line, their runtime-outputs are employed to measure the classification performance. For the reasoners which are used by a Java program via the OWL API, external time measurement is applied. All ontologies in the test suite are employed to compare the performance of the reasoners.

The experiments were performed under Linux 64-bit on a 4x AMD Opteron 8220 dual core, 2800 MHz CPU system with 16GB memory. For Java reasoners, Sun's Java Runtime Environment (JRE) version 1.6.0 was used with a Java HotSpot(TM) 64-Bit Server VM. We did not set a fixed maximum heap space but measured the minimum amount of heap space required to classify SNOMED CT in a separate experiment. The minimum required heap space has been approximated in steps of 0.5GB and the lowest heap space for which the reasoner classified SNOMED CT without crashing has been noted. All stated runtimes are averaged over 10 runs.

Classification performance As Table 5 shows, all tested reasoners succeeded in classifying SNOMED CT. For all input ontologies, CB took the least time to compute the subsumption hierarchy. FaCT++ and TrOWL REL are the only reasoners that are faster on NCI than on GO. FaCT++ needed longest to classify GO, RacerPro to classify the NCI ontology and HermiT to classify SNOMED CT (nearly 2 hours). The experiments measured only the classification time, and no loading and / or preprocessing times.

TBox consistency checking performance CB has no support for consistency checking. For reasoners that implement OWL API reasoning interfaces, the time that the call `reasoner.isConsistent()` took was

[16] http://www.ihtsdo.org/snomed-ct/
[17] http://code.google.com/p/cb-reasoner/
[18] http://lat.inf.tu-dresden.de/systems/cel/
[19] http://owl.man.ac.uk/factplusplus/
[20] http://www.hermit-reasoner.com/
[21] http://clarkparsia.com/pellet/
[22] http://www.racer-systems.com/
[23] http://research.ict.csiro.au/software/snorocket
[24] http://trowl.eu/

Table 5
Comparison of classification time in seconds

	CB	CEL	FaCT++	HermiT	Pellet	RP	SR	TR
GO	0.34	3.19	20.75	6.48	3.41	10.67	1.54	2.43
NCI	0.65	7.52	11.10	11.75	14.84	52.87	4.31	1.83
S CT	28.08	1,112.23	700.87	6,793.76	1,345.65	3,652.03	101.16	344.93

measured, before and after classification. Table 6 shows the duration of consistency checking. All stated times are before classification. After the first consistency check and after the classification, it is already known whether the ontology is consistent or not and the second consistency check takes less than one second for all tested reasoners. All input ontologies are consistent and all reasoners returned this result before and after classification.

The CEL manual states that an ontology must be classified before it can be checked whether the TBox is consistent. TrOWL REL also needs to classify first. When checking for consistency before classification, Snorocket outputs a warning that the ontology is not classified. All reasoners check for consistency rather fast. For tableau-based reasoners, this is probably due to the fact that for consistency checking they construct the model only once.

Concept satisfiability checking performance To determine the performance of concept satisfiability checking, we measure the time it takes each reasoner to check the satisfiability of each concept in the ontology, before and after classification. CB has no support for concept satisfiability checking. For reasoners that implement OWL API reasoner interfaces, we checked for concept satisfiability with the method `reasoner.isSatisfiable (concept)`. RacerPro does not really distinguish between TBox consistency and satisfiability. It offers the function `check-tbox-coherence` which returns a list of inconsistent / unsatisfiable atomic concepts. If the top concept occurs in this list, all concepts are unsatisfiable. It does not compute the concept hierarchy, so that it is much faster than to classify the TBox.

A more direct and comparable (with respect to RacerPro) way to retrieve unsatisfiable concepts via the OWL API would have been to call `reasoner.getInconsistentClasses()` for OWL API version 2 or respectively `reasoner.getUnsatisfiableClasses()` for the OWL API version 3, but as we wanted to measure the performance of concept satisfiability checking, we preferred to check the satisfiability of each single concept.

Table 6
Comparison of consistency checking time in seconds

	CB	CEL	FaCT++	HermiT	Pellet	RP	SR	TR
GO	–	2.17	0.36	0.00	0.27	–	0.00	0.00
NCI	–	0.65	0.71	0.00	0.38	–	0.00	0.00
SNOMED CT	–	0.88	15.3	0.00	16.78	–	0.00	0.00

Table 7 shows that the reasoners vary significantly in their runtimes. Some runtimes are so low that it can be assumed that they do not support the method, but compute satisfiability during classification and thus return reliable results only after the ontology is classified. For example, TrOWL REL checks after the classification whether the concept is subsumed by owl:Nothing.

Subsumption query performance To test subsumption query performance, we will query for all subclasses of the SNOMED CT concept "Fracture of lower limb", as presented in Fig. 2. We will test it in two ways: by querying for direct subclasses of its concept name `SCT_46866001` and by querying for direct subclasses of its anonymous class definition as stated in Fig. 2. The employed method of the OWL API is: `reasoner.getSubClasses()`, and the employed method of RacerPro `(tbox-retrieve (?x) (concept ?x has-child))`.

When performing this experiment, we noticed a correlation between the runtimes of the queries and the number of returned subclasses. Thus, Table 8 shows not only the runtimes but also the number of results. The four settings are tested sequentially in one run. CB does not support any subsumption querying. Snorocket returns a NullPointerException when the method is

Table 7
Comparison of concept satisfiability checking time in seconds

	CB	CEL	FaCT++	HermiT	Pellet	RP	SR	TR
GO BC	–	5.28	0.63	6.23	2.12	5.58	0.01	0.23
GO AC	–	2.08	0.07	0.06	0.12	0.00	0.01	0.03
NCI BC	–	4.46	1.26	11.73	3.47	37.73	0.01	0.06
NCI AC	–	3.19	0.14	0.09	0.18	0.00	0.01	0.04
S CT BC	–	38.42	22.37	5,276.85	56.91	273.45	0.07	5.17
S CT AC	–	34.59	1.76	1.36	6.07	0.00	0.06	0.46

BC stands for before classification and AC for after classification.

Table 8

Queries for subclasses of the SNOMED CT concept Fracture of lower limb: Comparison of subsumption query performance in seconds and number of results (i.e. returned subclasses)

	CB	CEL	FaCT++	HermiT	Pellet	RP	SR	TR
NC BC								
seconds	–	0.96	701.79	6,649.85	2,793.31	3,380.67	NPE	0.17
# results	–	1	20	20	20	20	–	0
AC BC								
seconds	–	0.00	0.06	16.94	0.49	0.74	NPE	0.00
# results	–	1	1	20	20	20	–	0
NC AC								
seconds	–	0.00	0.00	0.00	0.00	0.70	0.00	0.28
# results	–	20	20	20	20	20	20	20
AC AC								
seconds	–	0.00	0.06	17.12	0.00	0.92	6.97	0.00
# results	–	1	1	20	20	20	20	0

NC stands for named concept and AC for anonymous concept. BC stands for before classification and the second AC for after classification. NPE stands for NullPointerException.

called before classification. FaCT++, HermiT, Pellet and RacerPro classify the ontology when they receive the first query. The number of returned subclasses varies. HermiT, Pellet and RacerPro return all 20 subclasses in all settings. CEL returns the 20 subclasses only after classification and when the concept name is used. The other queries return owl:Nothing as the only result. FaCT++ returns the 20 subclasses when being queried for the concept name, while the query that contains the anonymous class definition returns the name of this concept.

Minimum heap space for Java reasoners Table 9 shows the minimum required amount of heap space for Java reasoners with SNOMED CT as input ontology. Memory exhaustion is a known problem in tableau-based reasoners when processing large ontologies, and our experiments confirm this. The minimum heap space is just an indicator and might vary for other systems. It needs to be pointed out that our experiments have been performed on a Java HotSpot(TM) 64-Bit Server VM. It is known that the 64-bit mode consumes around 30% more memory as the 32-bit mode, and it makes sense to use the 64-bit mode mostly if 4GB heap space is not sufficient in the 32-bit mode. Running a 32-bit JVM is not supported on the system we performed our tests on.

Soundness and completeness in practice In this paragraph, the computed concept hierarchies of the reasoners are analyzed to test whether the theoretical correctness of the tested reasoners can be confirmed in practice. For all included ontologies, we compared the output of each reasoner to the outputs of all other reasoners, with the rationale that if completely different reasoners generate the same output, the output is probably sound and complete. This does not exclude a scenario in which all reasoners output the same unsound statements and / or collectively do not produce inferences that should be produced. If the outputs differ from each other, we can infer that not all reasoners generate correct output. There is no standard specification on how to output the computed concept hierarchy, and thus we do not analyze the outputs line by line.

Table 9

Minimum heap space for Java reasoners

CB	CEL	FaCT++	HermiT	Pellet	RP	SR	TR
n/a	n/a	n/a	4.5	10	n/a	2.5	4

To summarize the insignificant differences that we found: In comparison to other reasoners, CB outputs less statements by omitting that top-level concepts are SubClassOf owl:Thing. CEL outputs additionally that every concept and every property is equivalent to itself. Also RacerPro generates additional axioms by stating for all leaf concepts (i.e. concepts that do not have subclasses) that they are superclasses of owl:Nothing. Apart from these differences and for SNOMED CT as input ontology, we found substantial differences: Pellet generates 386 inferences less than the other reasoners and 546 inferences that no other reasoner generates, while Snorocket misses 86 inferences that occur in the other outputs and generates 34 triples that no other reasoner generates. The missing and also the additional statements of Pellet and Snorocket have an empty intersection.

In the following, we will exploit the fact that SNOMED CT is released both in stated and in inferred form. The inferred form is the Relationships Table contained in the official SNOMED CT distribution and can be employed as gold standard to analyze computed concept hierarchies. The Stated Relationships Table differs from the Relationships Table in that it only contains those relationships that are directly asserted by authors or editors. When the generated OWL ontology is classified with a reasoner, the output should correspond to the Relationships Table.

As a first step, we successfully checked the accordance of the Stated Relationships Table and the generated OWL file. Then, to analyze the accordance of the computed concept hierarchies and the Relation-

Table 10
Missing / Additional inferred SubClassOf statements in regard to the Relationships Table

	CB	CEL	FaCT++	HermiT	Pellet	RP	SR	TR
Missing	0	0	0	0	386	0	86	0
Additional	0	0	0	0	546	0	34	0

Pellet:

Drug-induced immunodeficiency $\sqsubseteq$ *Drug-related disorder (missing)*
Biological substance poisoning $\sqsubseteq$ *Drug-related disorder (additional)*

Snorocket:

Amiloride + hydrochlorothiazide 2.5mg/25mg tablet $\sqsubseteq$ *Oral dosage form product (missing)*
Betaxolol hydrochloride 20mg tablet $\sqsubseteq$ *Oral dosage form product (additional)*

Fig. 3. Examples of missing and additional inferences with regard to the SNOMED CT Relationships Table.

ships Table, we compared the computed concept hierarchies to the Relationships Table, and the Relationships Table to the computed concept hierarchies. First, we compared all subclass rows from the Relationships Table that only include active concepts to the outputs of the tested reasoners. All outputs are missing 50 concept model attribute statements, which is caused by the way in which the OWL file is generated. All outputs except the one of CEL are missing 11 SubObjectPropertyOf statements. However, those statements are already present in the Stated Relationships Table and thus not really inferences. Pellet did not infer 386 SubClassOf relationships present in the Relationships Table and the Snorocket Protégé plugin did not infer 86 statements that are present in the Relationships Table. Table 10 summarizes the results. Both concept model attributes and SubObjectPropertyOf assertions are not counted in the table.

Finally, we checked for every inferred SubClassOf axiom of each of the outputs whether it exists in the Relationships Table to identify additional inferred statements. Ignoring tautological axioms such as the root node being SubClassOf owl:Thing and that owl:Nothing is SubClassOf all the leaf nodes, Pellet outputs 546 additional statements and Snorocket 34. Examples of missing and additional inferences are given in Fig. 3. With regard to the SNOMED CT Relationships Table, the outputs of all other included reasoners neither missed inferences, nor did they contain additional inferences.

As a result of this study we found that the Snorocket Protégé plugin did give correct results when run with Java 1.5 but produced some incorrect results when run with Java 1.6. The root cause of this problem has been fixed in the subsequent release (1.3.3). Furthermore, a new version (0.4.1) of CEL was released, which does not output the singletons for equivalent properties and concepts. The fix of Pellet's issue of missing and additional inferences will be part of its next release (i.e., 2.2.3). This shows that the analysis of computed concept hierarchies is valuable both to developers and to users of OWL reasoners. Another conclusion is that for SNOMED CT as input ontology, the comparison of the inferred concept hierarchies with each other delivered the same results as the comparison of the inferred concept hierarchies with the Relationships Table.

5.4. *Tradeoff between expressivity and classification performance*

Figure 4 shows the classification performance for SNOMED CT, with the reasoners ordered by increasing expressivity (as displayed on the x2-axis). Pellet and TrOWL REL are in the EL section because both of them support OWL 2 EL with implementations that are based on [1]. Reasoners within the same expressivity category are ordered alphabetically.

The time needed to classify SNOMED CT does not steadily rise with increasing expressivity. The very expressive reasoner FaCT++ is faster than CEL and Pellet. CB is the fastest reasoner even though it supports a more expressive language than OWL 2 EL.

6. Conclusion, discussion and future work

The main contribution of this paper is the definition of characteristics that are relevant to evaluate OWL reasoning engines in order to choose the most suitable reasoner for a given application, and the characterization of eight reasoning engines based on these properties. We showed that reasoners vary significantly with regard to all included characteristics. Therefore, a critical assessment and evaluation of core requirements is needed before selecting a reasoner for a real-life application. For example, let us consider a scenario in which a user chooses a reasoner for SNOMED CT. A crucial consideration is whether reasoning services such as incremental classification, rule support, justifications and ABox reasoning are required. Regarding practical usability, it needs to be decided which interfaces (OWL API, OWLlink, Jena) are needed, and whether the source of the reasoner should be open and

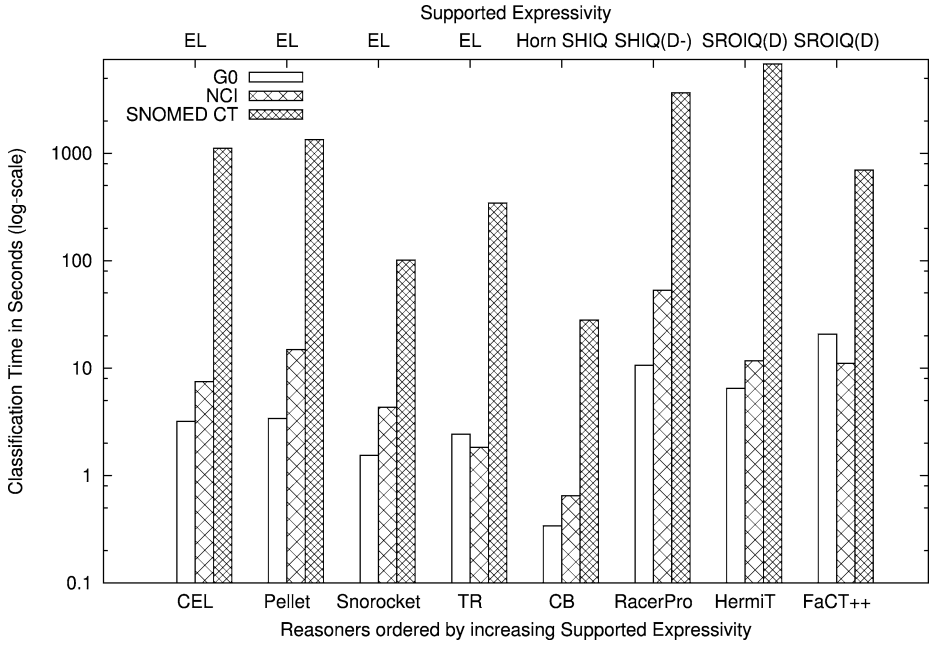

Fig. 4. Classification performance vs. supported expressivity.

come with a corresponding license. Also the platform on which the reasoner will run might influence the decision. With respect to the reasoner's performance, one of the aspects is how long the user is willing to wait for classification results, whether she wishes to query for anonymous concepts and concept satisfiability before classification.

Reasoning is an active field of research, and recent developments show that not only (the underlying methods of) established reasoners are being pushed further, but also new reasoners enter the field. Thus, this paper can only display a current snapshot of those rapid developments. Only dedicated OWL 2 EL and tableau-based reasoners have been taken into account for this comparison. Datalog engines, rule engines or reasoners that are based on theorem provers have not been included. The scope of this study is limited to ontologies in $\mathcal{EL}^+$. Performance results might be different for other ontologies, and when a reasoner is needed for a more expressive language, OWL 2 EL reasoners are not applicable. The selection of characteristics is not complete. Support for non-standard ontology features, such as description graphs, has not been included. Also, loading times or the different input and output formats that the reasoners can parse and write have not been evaluated. CB, for example, relies on the OWL functional syntax, while reasoners that integrate the OWL API are very flexible regarding serialization formats. Missing usability characteristics include support and documentation. Commercial reasoners generally offer more support, including support contracts. Also the level of documentation varies considerably for different reasoners. Regarding our experiments, it would have been fairer to measure subsumption checking performance for more than only one concept, as different reasoners might be optimized for different structures. Also, different reasoners are optimized for different settings of retrievals of direct / indirect subclasses / superclasses, so that all those scenarios should be included in a balanced comparison. The time it takes to retrieve inconsistent / unsatisfiable concepts would be another interesting experiment. Our experiments heavily rely on the OWL API, which contains functions that do not have a tightly specified functionality, and this might be the source of some of the variations of our obtained results. Furthermore, we did not include the supported OWL API version in the dimension practical usability. Future work includes measuring incremental reasoning performance, reasoning with more expressive ontologies, such as GALEN, and benchmarks that involve ABox reasoning as well as inconsistent ontologies and unsatisfiable concepts.

A positive outcome is that all eight tested reasoners succeed in classifying the very large ontology SNOMED CT. The advantage of this ontology is that it is released both in stated and in inferred form, so that

the concept hierarchies computed by the reasoners can not only be compared to each other but also to the inferred form, which can be employed as a gold standard to evaluate correctness. By comparing the outputs of the reasoners with each other and with the SNOMED CT Relationships Table, we found classification errors for Pellet and Snorocket that will be / have been fixed. Ongoing testing is necessary to evaluate the correctness of reasoners in practice. Our described characteristics can be applied to any reasoner and form a basis to evaluate reasoners not only on classification performance, but also on other aspects which can be relevant. We will present this study and future results on http://reasonerben.ch.

Acknowledgements

We like to thank the developers of all reasoners, especially (in alphabetical order of the corresponding reasoners) Yevgeny Kazakov, Julian Mendez and Boontawee Suntisrivaraporn, Dmitry Tsarkov, Boris Motik, Kendall Clark, Evren Sirin, Ralf Möller, Michael Wessel and Kay Hidde, Michael Lawley and Jeff Pan, for their active support, insights and interesting discussions. We also like to thank the reviewers for their constructive feedback.

References

[1] F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope. In *Proc. of the Nineteenth International Joint Conference on Artificial Intelligence*, Oct. 2005.

[2] F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope Further. In *Proc. of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.

[3] F. Baader, C. Lutz, and B. Suntisrivaraporn. Is Tractable Reasoning in Extensions of the Description Logic EL Useful in Practice? In *Proc. of the Methods for Modalities Workshop*, 2005.

[4] F. Baader, B. Ganter, B. Sertkaya, and U. Sattler. Completing description logic knowledge bases using formal concept analysis. In *IJCAI*, pages 230–235, 2007.

[5] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL – A Polynomial-time Reasoner for Life Science Ontologies. In *Proc. of the 3rd International Joint Conference on Automated Reasoning*, volume 4130, pages 287–291. Springer, 2006.

[6] F. Baader, C. Lutz, and A.-Y. Turhan. Small is Again Beautiful in Description Logics. *KI – Künstliche Intelligenz*, **24**(1):25–33, Feb. 2010.

[7] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2nd edition, 2007.

[8] S. Bail, B. Parsia, and U. Sattler. JustBench: A Framework for OWL Benchmarking. *The Semantic Web – ISWC 2010*, 2010.

[9] J. Bock, P. Haase, Q. Ji, and R. Volz. Benchmarking OWL reasoners. In *ARea2008 – Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*, 2008.

[10] S. Brandt. Polynomial Time Reasoning in a Description Logic with Existential Restrictions, GCI Axioms, and-What Else? In *ECAI*, volume 16, pages 298–302, 2004.

[11] J. Broekstra, A. Kampman, and F. Van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. *The Semantic Web – ISWC 2002*, volume 2342, pages 54–68, 2002.

[12] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations, 2004.

[13] R. Cornet and N. de Keizer. *Forty years of SNOMED: A literature review*. BMC medical informatics and decision making, 2008.

[14] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in Description Logics. *Principles of Knowledge Representation*, pages 191–236, 1996.

[15] T. Gardiner, D. Tsarkov, and I. Horrocks. Framework for an automated comparison of description logic reasoners. In *The Semantic Web – ISWC 2006*, volume 4273, pages 654–667. Springer, 2006.

[16] B. Grau, I. Horrocks, and Y. Kazakov. Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence*, **31**:273–318, 2008.

[17] Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, **3**(2–3):158–182, 2005.

[18] V. Haarslev, R. Möller, and S. Wandelt. The revival of structural subsumption in tableau-based description logic reasoners. In *Proc. of the 2008 International Workshop on Description Logics*, 2008.

[19] V. Haarslev and R. Müller. RACER System Description. *Automated Reasoning*, **2083**:701–705, 2001.

[20] M. Horridge and S. Bechhofer. The OWL API: A Java API for Working with OWL 2 Ontologies. In *6th OWL Experienced and Directions Workshop*, 2009.

[21] M. Horridge, B. Parsia, and U. Sattler. Justification Oriented Proofs in OWL. In *The Semantic Web – ISWC 2010*, volume 6496, pages 354–369. Springer, 2010.

[22] I. Horrocks. *Optimising tableaux decision procedures for description logics*. PhD thesis, University of Manchester, 1997.

[23] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, **8**(3):239–264, 2000.

[24] Y. Kazakov. SRIQ and SROIQ are Harder than SHOIQ. In *Proc. of the 21st International Workshop on Description Logics*, 2008.

[25] Y. Kazakov. Consequence-Driven Reasoning for Horn SHIQ Ontologies. In *Proc. of the 21st International Conference on Artificial Intelligence*, pages 2040–2045, 2009.

[26] A. Kiryakov, D. Ognyanov, and D. Manov. OWLIM – A Pragmatic Semantic Repository for OWL. In *Web Information Systems Engineering Workshops*, pages 182–192. Springer, 2005.

[27] B. Konev, C. Lutz, D. Walther, and F. Wolter. Formal Properties of Modularisation. *Modular Ontologies*, **5445**:25–66, 2009.

[28] M. Krötzsch, S. Rudolph, and P. Hitzler. Description Logic Rules. In *Proc. of the 18th European Conference on Artificial Intelligenc*, pages 80–84. IOS Press, 2008.

[29] M. Krötzsch, S. Rudolph, and P. Hitzler. ELP: Tractable Rules for OWL 2. In *The Semantic Web – ISWC 2008*, pages 649–664. Springer, 2008.

[30] M. Lawley and C. Bousquet. Fast Classification in Protege: Snorocket as an OWL2 EL Reasoner. In *Australasian Ontology Workshop*, 2010.

[31] T. Liebig. Reasoning with OWL – System Support and Insights, 2006.

[32] T. Liebig, M. Luther, O. Noppens, M. Rodriguez, D. Calvanese, M. Wessel, R. Möller, M. Horridge, S. Bechhofer, D. Tsarkov et al. OWLlink: DIG for OWL 2. In *5th OWL Experienced and Directions Workshop*, 2008.

[33] M. Luther, T. Liebig, S. Böhm, and O. Noppens. Who the Heck Is the Father of Bob? In *6th European Semantic Web Conference*, pages 66–80, 2009.

[34] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a Complete OWL Ontology Benchmark. *The Semantic Web: Research and Applications*, **4011**:125–139, 2006.

[35] J. Mendez and B. Suntisrivaraporn. Reintroducing CEL as an OWL 2 EL Reasoner. In *Proc. of the 2009 International Workshop on Description Logics*, volume 477, 2009.

[36] R. Mishra and S. Kumar. Semantic web reasoners and languages. *Artificial Intelligence Review*, **35**:339–368, 2010.

[37] B. Motik, R. Shearer, and I. Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In *Proc. of the 21st International Conference on Automated Deduction*, volume 4603, pages 67–83. Springer, 2007.

[38] B. Motik and R. Studer. KAON2 – A Scalable Reasoning Tool for the Semantic Web. In *Proc. of the 2nd European Semantic Web Conference*, 2005.

[39] B. Parsia, C. Halaschek-Wiener, and E. Sirin. Towards Incremental Reasoning Through Updates in OWL-DL. In *Reasoning on the Web*, 2006.

[40] Y. Ren, J. Pan, and Y. Zhao. Soundness Preserving Approximation for TBox Reasoning. In *Proc. of the 25th AAAI Conference*, 2010.

[41] A. Riazanov and A. Voronkov. *Vampire 1.1*. Automated Reasoning, 2001.

[42] S. Rudolph, T. Tserendorj, and P. Hitzler. What Is Approximate Reasoning? *Web Reasoning and Rule Systems*, **5341**:150–164, 2008.

[43] R. Shearer, B. Motik, and I. Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *5th OWL Experienced and Directions Workshop*, 2008.

[44] E. Sirin and B. Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *3rd OWL Experiences and Directions Workshop*, volume 4, 2007.

[45] E. Sirin, B. Parsia, B. Grau, a. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, **5**(2):51–53, June 2007.

[46] H. Stuckenschmidt, C. Parent and S. Spaccapietra. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*. Springer, 2009.

[47] B. Suntisrivaraporn. Empirical evaluation of reasoning in lightweight DLs on life science ontologies. In *Proc. of the 2nd Mahasarakham International Workshop on AI*, 2008.

[48] B. Suntisrivaraporn. Module Extraction and Incremental Classification: a Pragmatic Approach for EL+ Ontologies. In *Proc. of the 5th European Semantic Web Conference*, pages 230–244. Springer, 2008.

[49] B. Suntisrivaraporn. *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies*. PhD thesis, TU Dresden, 2009.

[50] E. Thomas, J. Pan, and Y. Ren. TrOWL: Tractable OWL 2 Reasoning Infrastructure. In *Proc. of the Extended Semantic Web Conference*. Springer, 2010.

[51] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.

[52] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Third International Joint Conference on Automated Reasoning*, pages 292–297. Springer, 2006.

[53] A. Turhan, S. Bechhofer, A. Kaplunova, T. Liebig, M. Luther, R. Möller, O. Noppens, P. Patel-Schneider, B. Suntisrivaraporn, and T. Weithöner. DIG 2.0 – Towards a Flexible Interface for Description Logic Reasoners. In *Proc. of the OWL Experiences and Directions Workshop at the ISWC*, volume 6. Citeseer, 2006.

[54] T. Weithöner, T. Liebig, M. Luther, S. Böhm, F. Von Henke, and O. Noppens. Real-World Reasoning with OWL. *The Semantic Web: Research and Applications*, **4519**:296–310, 2007.