JustBench: A Framework for OWL Benchmarking

Samantha Bail, Bijan Parsia, Ulrike Sattler

The University of Manchester
Oxford Road, Manchester, M13 9PL
{bails,bparsia,sattler@cs.man.ac.uk}

Abstract. Analysing the performance of OWL reasoners on expressive OWL ontologies is an ongoing challenge. In this paper, we present a new approach to performance analysis based on *justifications* for entailments of OWL ontologies. Justifications are minimal subsets of an ontology that are sufficient for an entailment to hold, and are commonly used to debug OWL ontologies. In *JustBench*, justifications form the key unit of test, which means that *individual justifications* are tested for correctness and reasoner performance instead of entire ontologies or random subsets. Justifications are generally small and relatively easy to analyse, which makes them very suitable for transparent analytic micro-benchmarks. Furthermore, the JustBench approach also allows us to isolate reasoner errors and inconsistent behaviour. We present the results of initial experiments using JustBench with FaCT++, HermiT, and Pellet. Finally, we show how JustBench can be used by reasoner developers and ontology engineers seeking to understand and improve the performance characteristics of reasoners and ontologies.

1 Introduction

The Web Ontology Language (OWL) notoriously has very bad worse case complexity for key inference problems, at least, OWL Lite (EXPTIME-complete for satisfiability), OWL DL 1 & 2 (NEXPTIME-complete), and OWL Full (undecidable) (see [5] for an overview). While there are several highly optimised reasoners (FaCT++, HermiT, KAON2, Pellet, and Racer) for the NEXPTIME logics, it remains the case that it is frustratingly easy for ontology developers to get unacceptable or unpredictable performance from them on their ontologies. Reasoner developers continually tune their reasoners to user needs in order to remain competitive with other reasoners. However, communication between reasoner developers and users is tricky and, especially on the user side, often mystifying and unsatisfying.

Practical OWL DL reasoners are significantly complex pieces of software, even just considering the core satisfiability testing engine. The basic calculi underlying them are daunting given that they involve over a dozen inference rules with complex conditions to ensure termination. Add in the extensive set of optimisations and it is quite difficult for non-active reasoner developers to have a

reasonable mental model of the behaviour of reasoners. Middleware issues introduce additional layers of complexity ranging from further optimisations (for example, classification vs. isolated subsumption tests) to the surprising effects of different parsers on system performance.

In this paper, we present a new approach to analysing the behaviour of reasoners by focusing on justifications of entailments. Justifications—minimal entailing subsets of an ontology—already play a key role in debugging unwanted entailments, and thus are reasonably familiar to users. They are small and clearly defined subsets of the ontology that can be analysed manually if necessary, which reduces user effort when attempting to understand the source of an error in the ontology or unwanted reasoner behaviour. We present results from analysing six ontologies and three reasoners and argue that justifications provide a reasonable starting point for developing empirically-driven analytical micro-benchmarks.

2 Reasoner Behaviour Analysis

2.1 Approaches to Understanding Reasoner Behaviour

Consider five approaches to understanding the behaviour of reasoners on a given ontology, especially by ontology modellers:

1. **Training** In addition to the challenges of promulgating detailed understanding of the performance implications of the suite of calculi and associated optimisations (remembering that new calculi or variants thereof are cropping up all the time), it is unrealistic to expect even sophisticated users to master the engineering issues in particular implementations. Furthermore, it is not clear that the requisite knowledge is available to be disseminated: New ontologies easily raise new performance issues which require substantial fundamental research to resolve.
2. **Tractable logics** In recent years, there has been a renaissance in the field of tractable description logics which is reflected in the recent set of tractable OWL 2 profiles.[1] These logics tend to not only have good worst case behaviour but to be "robust" in their performance profile especially with regard to scalability. While a reasonable choice for many applications, they gain their performance benefits by sacrificing expressivity which might be required.
3. **Approximation** Another approach is to give up on soundness or completeness when one or the other is not strictly required by an application, or, in general, when some result is better than nothing. Approximation [17, 3, 16] can either be external (e.g., a tool which takes as input an OWL DL ontology and produces an approximate OWL EL ontology) or internal (e.g., anytime computation or more sophisticated profile approximation). A notable difficulty of approximation approaches is that they require *more* sophistication on the part of users and sophistication of a new kind. In particular, they

[1] http://www.w3.org/TR/2009/REC-owl2-profiles-20091027

need to understand the semantic implications of the approximation. For example, it would be quite simple to make existing reasoners return partial results for classification—classification is inherently anytime. But then users must recognise that the absence of an entailment no longer reliably indicates non-entailment. In certain UIs (such as the ubiquitous tree representations), it is difficult to represent this additional state.

4. **Fixed rules of thumb** These may occur as a variant or result of training or be embodied in so-called "lint" tools [12]. The latter is to be much preferred as such tools can evolve as reasoners do, whereas "folk knowledge" often changes slowly or promulgates misunderstanding. For example, the rules of thumb "inverses are hard" and "open world negation is less efficient than negation as failure"[2] do not help a user determine which (if either) is causing problems in their particular ontology/reasoner combination. This leads users to start ripping out axioms with the "dangerous" constructs in them which, e.g., for negation in the form of disjointness axioms, may in fact make things worse. Lint tools fare better in this case but do not support *exploration* of the behaviour of a reasoner/ontology combination, especially when one or the other does not fall under the lint tools coverage. Finally, rules of thumb lead to *manual* approximation which can distort modelling.
5. **Analytical tools** The major families of analytical tools are profilers and benchmarks. Obviously, one can use standard software profilers to analyse reasoner/ontology behaviour, and since many current reasoners are open source, one can do quite well here. This, however, requires a level of sophistication with programming and specific code bases that is unreasonable to demand of most users. While there has been some work on OWL specific profilers [19], there are none, to our knowledge, under active development. Benchmarks, additionally, provide a common target for reasoner developers to work for, hopefully leading to convergence in behaviour. On the flip side, benchmarks cannot cover all cases and excessive "benchmark tuning" can inflate reasoner performance with respect to the benchmarks without improving general behaviour in real cases.

2.2 Benchmarks

Application and Analytical Benchmarks For our current purposes, a *benchmark* is simply a reasoning problem, typically consisting of an ontology and an associated entailment. A *benchmark suite*, although often called a benchmark or benchmarks, is a set of benchmarks.

We can distinguish benchmark suites by three characteristics: their *focus*, their *realism*, and their *method of generation*. With regard to focus, the classic distinction is between *analytical* benchmarks and *application* benchmarks.

Analytical benchmarks attempt to determine the presence or absence of certain performance related features, e.g., the presence of a query optimiser in a

[2] This latter rule of thumb is actually *false* in general. Non-monotonic features generally *increase* worst case complexity, often quite significantly.

relational database can be detected[3] by testing a query written in sub-optimal form. More generally, they attempt to isolate particular behaviours of the system being analysed.

Application benchmarks attempt to predict the behaviour of a system on certain classes of application by testing an example (or select examples) of that class. The simplest form of an application benchmarking is retrospective recording of the behaviour of the application on the system in question in real deployment (i.e., performance measurement). Analytical benchmarks aim to provide a more precise understanding of the tested system, but that precision may not help predict how the system will perform in production. After all, an analytical benchmark does not say which part of the system will be stressed by any given application. Application benchmarks aim for better predictions of actual behaviour in production, but often this is at the expense of understanding. Accidental or irrelevant features might dominate the benchmark, or the example application may not be sufficiently representative.

In both cases, benchmark suites might target particular classes of problem, for example, conjunctive query answering at scale in the presence of $\mathcal{SHIQ}$ TBoxes.

Choice of Reasoning Problems In order to be reasonably analytic, benchmarks need to be understandable enough so that the investigator can correlate the benchmark and features thereof with the behaviour observed either on theoretical grounds, e.g., the selectivity of a query, or by experimentation, e.g. by making small modifications to the test and observing the result. If we have a good theoretical understanding, then individual benchmarks need not be small. However, we do not have a good theoretical understanding of the behaviour of reasoners on real ontologies and, worse, real ontologies tend to be extremely heterogenous in structure, which makes sensible uniform global modifications rather difficult. While we we can measure and compare the performance of reasoners on real ontologies, we often cannot understand or analyse *why* some (parts of) ontologies are particularly hard for a certain reasoner—or even isolate these parts. Thus, we turn to subsets of existing ontologies. However, arbitrary subsets of an ontology are unlikely to be informative and there are too many for a systematic exploration of them all. Thus, we need a selection principle for subsets of the ontology. In JustBench, our initial selection principle is to select *justifications* of atomic subsumptions, which will be discussed in section 3.

Artificial subsets Realism forms an axis with completely artificial problems at one pole, and naturally occurring examples at the other. The classic example of an artificial problem is the kSAT problem for propositional, modal, and description logics [7, 18, 10, 11]. kSAT benchmark suites are presented in terms of how to generate random formulae (to test for satisfiability) according to certain parameters. Some of the parameters can be fixed for any test situation (e.g.

[3] The retrospective on the Wisconsin Benchmark [6] for relational databases has a good discussion of this.

clause length which is typically 3) and others are allowed to vary within bounds. Such benchmarks are comparatively easy to analyse theoretically[4] as well as empirically.

However, these problems may not even *look* like real problems (kSAT formulae have no recognisable subsumption or equivalence axioms), so extrapolating from one to the other is quite difficult. One can always use naturally occurring ontologies when available, but they introduce many confounding factors. This includes the fact that users tend to modify their ontologies to perform well on their reasoner of choice. Furthermore, it is not clear that existing ontologies will resemble future ontologies in useful ways. This is especially difficult in the case of OWL DL due to the fragility of reasoner behaviour: seemingly innocuous changes can have severe performance effects. Also, for some purposes, existing ontologies are not hugely useful—for example, for determining scalability, as existing ontologies can only test for scalability up to their actual size.

The realism of a benchmark suite can constrain its method of generation. While artificial problems (in general) can be hand built or generated by a program, naturally occurring examples have to be found (with the exception of naturally occurring examples which are generated e.g., from text or by reduction of some other problem to OWL). Similarly, application benchmarks must be at least "realistic" in order to be remotely useful for predicting system behaviour on real applications.

Modules A module is a subset of an ontology which captures "everything" an ontology has to say about a particular subsignature of the ontology [4], that is, a subset which entails everything that the whole ontology entails which can be expressed in the signature of the module itself. Modules are attractive for a number of reasons including the fact that they capture all the relevant entailments and support a principled removal of "slow" parts of an ontology. However, most existing accounts of modularity are very fine grained with respect to signature choice, which preclude blind examination of all modules of an ontology.

If we restrict attention to modules for the signature of an atomic subsumption (which corresponds more closely to justifications for atomic subsumptions) we find that modules can be too big. First, at least by current methods, modules contain all justifications for *all* entailments expressible in their signature. As we can see in the Not-Galen ontology, this can lead to very large sets even just considering one subsumption. Second, current and prospective techniques involve various sorts of approximation which brings in additional axioms. While this excess is reasonable for many purposes, and might be more realistic as a model for a stand alone ontology, it interferes with the analysability of the derived benchmark. That being said, modules clearly have several potential roles for benchmarking, and incorporating them into *JustBench* is part of our future work.

[4] "Easy" in the sense of possible and feasible enough that analyses eventually emerge.

2.3 Existing OWL Benchmarks

The most popular reasoner benchmark, at least in terms of citation count, is the Lehigh University Benchmark (LUBM) [9]. LUBM is designed primarily to test the scalability of conjunctive query and consists of a small, simple, hand-built "realistic" ontology, a program for generating data conforming to that ontology, and a set of 14 hand-built "realistic" queries. LUBM is an application focused, realistic benchmark suite with artificial generation. LUBM's ontology and data were notoriously weak, for example, the ontology lacked coverage of many OWL features, a fact that the University Ontology Benchmark (UOBM) [13] was invented to rectify. For an extensive discussion and critique of existing synthetic OWL benchmarks see [20].

Several benchmarks suites, notable those described in [14, 8], make use of naturally occurring ontologies, but do not attempt fine grained analysis of how the reasoners and ontologies interact. Generally, it can be argued that the area of transparent micro-benchmarks based on real (subsets of) OWL ontologies, as opposed to comprehensive (scalabiliy-, system-, or application) benchmarks is currently neglected.

3 Justification-Based Reasoner Benchmarking

Our goal is to develop a framework for benchmarking ontology TBoxes which is analytic, uses real ontologies, and supports the generation of problems. In order to be reasonably analytic, particular benchmarks need to be understandable enough so that the investigator can correlate the benchmark and features thereof with the behaviour observed either on theoretical grounds, e.g., the selectivity of a query, or by experimentation, e.g., by making small modifications to the test and observing the result. If we have a good theoretical understanding, then individual benchmarks need not be small. However, we do not have a good theoretical understanding of the behaviour of reasoners on (arbitrary) real ontologies and, worse, real ontologies tend to be extremely heterogenous in structure, which makes sensible uniform global modifications rather difficult. Thus, we turn to subsets of existing ontologies. However, arbitrary subsets of an ontology are unlikely to be informative and there are too many for a systematic exploration of them all. Thus, we need a selection principle for subsets of the ontology. In JustBench, our initial selection principle is to select *justifications* of entailments, e.g., of atomic subsumptions.

Definition 1 (Justification) A set of axioms $J \subseteq \mathcal{O}$ is a justification for $\mathcal{O} \models \eta$ if $J \models \eta$ and, for all $J' \subset J$, it holds that $J' \not\models \eta$.

As an example, the following ontology[5] entails the atomic subsumption *C SubClassOf: owl:Nothing*, but only the first three axioms are necessary for the

[5] We use the Manchester OWL Syntax for all examples, omitting auxiliary declarations of entities for space and legibility reasons.

entailment to hold. Therefore, the set {*C SubClassOf: A and D, A SubClassOf: E and B, B SubClassOf not D and r some D*} is a justification for this entailment.

O = { *C SubClassOf: A and D,*
A SubClassOf: E and B,
B SubClassOf: not D and r some D,
F SubClassOf: r only A,
D SubClassOf: s some owl:Thing }

The size of a justification can range, in principle, from a single axiom to the number of all axioms in the ontology, with, in one study, an average of approximately 2 axioms per justification [1]. The number of justifications for an entailment can be exponential in the size of the ontology, and multiple (potentially overlapping) justifications for a single entailment occur frequently in ontologies used in practice.

An explanation framework that provides methods to exhaustively compute all justifications for a given entailment has been developed for the OWL API v3,[6] which we use in our benchmarking framework.

3.1 Limitations of this selection method

Justifications, while having several attractive features as benchmarks, also have drawbacks including: First, we can only generate test sets if computing at least some of the entailments and at least some of their justifications for them is feasible with at least one reasoner. Choice of entailment is critical as well, although, on the one hand, we have a standard set of entailments (atomic subsumptions, instantiations, etc.) and on the other hand we can analyse arbitrary sets of entailments (e.g., conjunctive queries derived from an application). As the test cases are generated by a reasoner, their correctness is determined by the correctness of the reasoner, which itself is often at issue. This problem is mitigated by checking individual justifications on all reasoners (for soundness) and using different reasoners to generate all entailments and their justifications (for completeness). The latter is very time consuming and infeasible for some ontologies.

Second, justification-based tests do not test scalability, nor do they test interactions between unrelated axioms, nor do they easily test non-entailment finding, nor do they test other global effects. With regard to scalability, we have two points: 1) Not every performance analysis needs to tackle scalability. For example, even if a reasoner can handle an ontology (thus, it scales to that ontology), its performance might be less than ideal. 2) Analysis of scalability problems needs to distinguish between reasoner weaknesses that are merely due to scale and those that are not. For example, if a reasoner cannot handle a particular two line ontology, it will not be able to handle that ontology with an additional 400 axioms. Thus, micro-benchmarks are still useful even if scalability is not relevant.

[6] http://owlapi.sourceforge.net

Finally, in the first instance, justification test successful entailment finding, but much of what an OWL reasoner does is find non-entailments. Non-entailment testing is a difficult matter to support analytically, however, even their justifications offer some promise. For example, we could work with repaired justifications.

3.2 JustBench: System Description

The *JustBench* framework is built in Java using the OWL API v3 and consists of two main modules that generate the justifications for an ontology and perform the benchmarks respectively. The generator loads an ontology from the input directory, finds entailments using the *InferredOntologyGenerator* class of the OWL API and generates justifications for these entailments with the explanation interface. The entailments in question are by adding specific *InferredAxiomGenerator*s to the ontology generator. For example, one can add *InferredSubClassAxiomGenerator* to get all subsumptions between named classes and *InferredClassAssertionAxiomGenerator* to get all atomic instantiations. By default, we just look for atomic subsumptions and unsatisfiable classes. The justifications and entailments are saved in individual OWL files which makes them ready for further processing by the benchmarking module.

For each performance measurement, a new instance of the OWLReasoner class is created which loads the justification and checks whether it entails the subsumption saved as *SubClassOf* axiom in the entailment ontology. We measure the times to create the reasoner and load the ontology, the entailment check using the *isEntailed()* call to the respective reasoner, and the removal of the reasoner instance with *dispose()*. Regarding the small run-times of the entailment checks, there exists a trade-off between fine-grained, transparent micro-benchmarks and large test cases, where the results may be more robust to interference, but also harder to interpret for users. Limiting the impact that actions in the Java runtime and the operating system have on the measurements is an important issue when benchmarking software performance [2], which we take into account in our framework. In order to minimise measurement variation, the sequence of load, check, dispose is repeated a large number of times (1000 in our current setting) and the median of the values measured after a warm-up phase is taken as the final result. In preliminary tests it was detected that the mean value of the measurements was distorted due to a small number of outliers that differed from the majority of the measured values by several orders of magnitude, which was presumably caused by the JVM garbage collection. Basing the measurement on the median instead proved to yield stable and more reliable results.

We also experimented with a slightly different test involving a one-off call to *prepareReasoner()* is included before the measured entailment check. *prepareReasoner()* triggers a complete classification of the justification. Thus, we can isolate the time required to do a simple "lookup" for the atomic subsumption in the entailment. The times for loading, entailment checking and disposing are then saved along with the results of the entailment checks. Since the tested ontologies are justifications for the individual entailments, this should naturally

return *true* for all entailment checks if the reasoner works correctly. As we will show in the next section, a *false* result here can indicate a reasoner error.

4 Experiments and Evaluation

4.1 Experimental Setup

The test sets were generated using JustBench and FaCT++ 1.4.0 on a Mac Pro desktop system (2.66 GHz Dual-Core Intel Xeon processor, 16 GB physical RAM) with 2GB of memory allocated to the Java virtual machine. The tested ontologies were Building, Chemical, Not-Galen (a modified version of the Galen ontology), DOLCE Lite, Wine and MiniTambis.[7] This small test set can already be regarded as sufficient to demonstrate our approach and show how its transparency and restriction to small subsets of the ontologies helps to isolate and understand reasoner behavior, as well as quickly trace the sources of errors.

Most test sets could be generated by our system within a few minutes, however, for the Not-Galen ontology the process was aborted after it had generated several hundred explanations for a single entailment. In order to limit the processing time, a reasoner time out was introduced, as well as a restriction on the number of justifications to be generated. Thus, the justifications for Not-Galen are not complete, and we assume that generating all explanations for all entailments of this particular ontology is not feasible in practical time. The number of justifications for each entailment ranged from 1 to over 300, as in the case of Not-Galen, with the largest containing 36 axioms.

The benchmarking was performed on the same system as the test set generation using three reasoners that are currently compatible with the OWL API version 3, namely FaCT++ 1.4.0, HermiT 1.2.3, and Pellet 2.0.1.

4.2 Results and Discussion

Reasoner Performance The measurements for the justifications generated from our five test ontologies show a clear trend regarding the reasoner performance. Generally, it has to be noted that the performance of all three reasoners can be regarded as suitably on these ontolgoies, and there are no obvious hard test cases in this test set. On average, FaCT++ consistently performs best in almost all checks, with HermiT being slowest in most cases. Pellet exhibits surprising behaviour, as it starts out with a performance close to that of FaCT++ for smaller justifications and then approximates or even "overtakes" HermiT for justifications with a larger number of axioms. This behaviour, e.g., as shown in figure 1, is seen in all the ontologies tested.

Generally, the time required for an entailment check grows with the size of the justification for all three reasoners, as shown in figure 2—justifications with a size larger than 13 are all obtained from the Not-Galen ontology. Again, Pellet

[7] All ontologies that were used in the experiments may be found online: http://owl.cs.man.ac.uk/explanation/justbenchmarks.

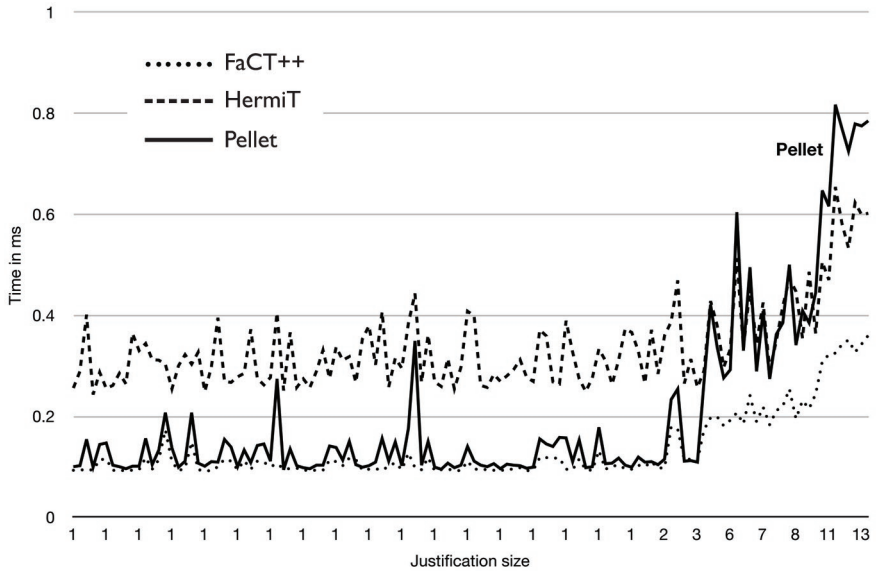

Fig. 1. Reasoner performance on justifications of the MiniTambis ontology

exhibits the behaviour mentioned above and eventually "overtakes" HermiT. The dip at size 16 is caused by the existence of only one justification of this size and can be neglected here.

HermiT in particular starts out with a higher baseline than the other reasoners, but only increases slowly with growing justification size. We are investigating further to pinpoint the exact factors in play.

For the atomic subsumptions in our examples, the expressivity—which seems quite wide ranging—does not significantly affect performance. The average time for each size group indicates that the hardest expressivities for all reasoners are $\mathcal{ALCN}$ and $\mathcal{ALCON}$. We expect that a analysis of the laconic versions of these justifications i.e., that only contain the axiom parts that are relevant to the entailment) will reveal to which extent the performance is affected by the use of expressive constructors.

Reasoner Errors While the system returns *true* for nearly all entailment checks, a small subset of Not-Galen is wrongly identified as not entailed by Pellet after adding a call to *prepareReasoner()* to force a full classification of the justification. All the misclassified justifications have the DL expressivity $\mathcal{ALEH}$, indicating that they contain subproperty hierarchies. On closer inspection it can be found that Pellet produces an error for the justifications that have axioms of the form

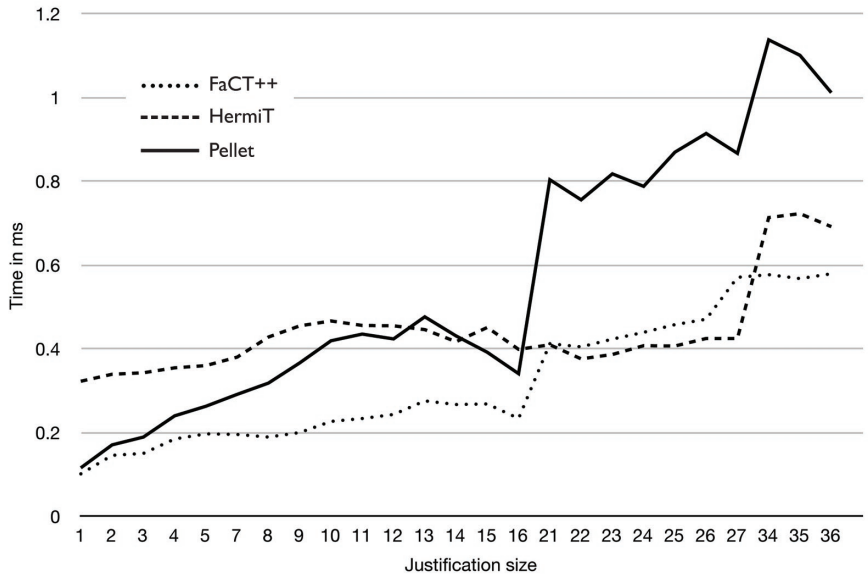

Fig. 2. Performance of reasoners depending on size of justifications

Tears SubClassOf:
NAMEDBodySubstance, isActedOnSpecificallyBy some
(Secretion and (isFunctionOf some LachrymalGland))

where *isActedOnSpecificallyBy* is a subproperty of some other property that is necessary for the entailment to hold. Communication with Pellet developers revealed that the problem is due to an error in the optimizations of the classification process and not in the core satisfiability check. This demonstrates the need to test all code paths.

By using justifications for the testing process, we detected and isolated and error which affects the correctness of the reasoner but was not otherwise visible. Performing an entailment check on the whole ontology would not exhibit this behaviour, as several other justifications for the entailment masked the entailment failure.

Errors Caused by Signature Handling We also identified a problem in how FaCT++ handles missing class declarations when performing entailment checks. For some justifications FaCT++ aborts the program execution with an "invalid memory access error", which is not shown by Pellet and HermiT. We isolated the erroneous justifications and perform entailment checks outside the benchmarking framework to verify that the problem was not caused by any of the additional calls to the OWL API made by JustBench. We found that the subsumptions were

all entailed due to the superclass being equivalent to *owl:Thing* in the ontology. Consider the following entailment:

NerveAgentSpecificPublishedWork
  SubClassOf: PublishedWork

and the justification for it consists of the following three axioms:

refersToPrecursor
  Domain: PublishedWork

NerveAgentRelatedPublishedWork
  SubClassOf: PublishedWork

VR_RelatedPublishedWork
  EquivalentTo: refersToPrecursor only VR_Precursor
  SubClassOf: NerveAgentRelatedPublishedWork

The subclass *NerveAgentSpecificPublishedWork* does not occur in the justification, as the entailment follows from *Class: PublishedWork EquivalentTo: owl:Thing* and therefore the subclass would not be declared in the justification. How should an OWL reasoner handle this case? Pellet and HermiT accept the ontologies and verify the entailment, whereas FaCT++ requires the signature of the entailment to be a subset of the signature of the justification. This causes FaCT++ to not even perform an entailment check and abort with the error "Unable to register 'NerveAgentSpecificPublishedWork' as a concept". While this is not a correctness bug per se, it is a subtle interoperability issue.

4.3 Additional Tests and Discussion

Performance for Full Classification In order to compare our justification-based approach to typical benchmarking methods, we measure a full classification of each of our test ontologies. Therefore, an instance of the OWL API's *InferredOntologyGenerator* class is generated and the time required for a call to its *fillOntology()* method is measured to retrieve all inferred atomic subsumptions from the respective ontology. Surprisingly, the rankings based on the individual justifications are inverted here: FaCT++ performed worst for all tested ontologies (except for Wine, where the reasoner cannot handle a "PositiveInteger" datatype and crashes), with an average of 1.84 s to retrieve all atomic subsumptions. HermiT and Pellet do this form of classification in much shorter time (0.98 s and 1.16 s respectively), but the loading times for HermiT (a call to *createReasoner()*) are an order of magnitude larger than those of FaCT++.

Additional Entailments Choice of entailments makes a big difference to the analysis of the ontology. For example, we examined an additional ontology which

has a substantial number of individuals. For full classification of the ontology, both HermiT and Pellet performed significantly worse than FaCT++. Using JustBench with justifications for all entailed atomic subsumptions of the ontology did not lead to any explanation for this behaviour: all three reasoners performed well on the justifications and the sum of their justification reasoning times was much less than their classification time. However, after adding the justifications for inferred class assertions to the test set, the time HermiT takes for entailment checks for these justifications is an order of magnitude larger than for the other reasoners. The isolation of the classes of entailment, as well as shared characteristics of entailments in each class, falls out quite naturally by looking at their justification. In this case, it is evident that there is a specific issue with HermiT's instantiation reasoning and we have small test cases to examine and compare with each other.

An Artificially Generated Ontology In an additional test with an artificially generated ontology we attempt to verify our claim about loading and classification times of the three reasoners. The ontology contains over 200 subsumptions of the form

A1EquivalentTo: A2
and (p some (not (A2)))

with entailments being atomic subsumptions of the type *A1 SubClassOf: A2*, *A2 SubClassOf: A3* ... *A1 SubClassOf: A210*. Justifications for 62 of these entailments were generated before the system ran out of memory, and the entailments were checked against their respective justifications and the full ontology. The right chart of figure 3 shows clearly how the performance of both Pellet and FaCT++ for an entailment check worsens with growing ontology size, whereas HermiT has an almost consistently flat curve. FaCT++ in particular shows almost exponential growth. In contrast, the loading times for larger ontologies only grow minimally for Pellet and FaCT++, while HermiT's loading time increases rapidly, as can be seen in the left chart of figure 3.

All three reasoners perform much worse on the artificial ontology than on the "real-life" ones (except for Not-Galen). This is a bit surprising, considering that the expressivity of this ontology is only $\mathcal{ALC}$, as opposed to the more expressive $\mathcal{ALCF}(\mathcal{D})$, $\mathcal{SHIF}$, $\mathcal{SHOIN}(\mathcal{D})$, and $\mathcal{ALCN}$ respectively of the other ontologies. The justifications for its entailments however are disproportionally large (up to 209 axioms for *Class: A1 SubClassOf: A210*) whereas those occurring in the other "real" ontologies have a maximum size of only 13 axioms. This indicates that a complex justificatory structure with a large number of axioms in the justifications poses a more difficult challenge for the reasoners.

The measurements based on the artificial ontology indicate that HermiT performs more preparations in its *createReasoner()* method and has only minimal lookup times, which confirms our results from the entailment checks following a call to *prepareReasoner()*. We can conclude that, once the ontology is loaded and fully classified, HermiT performs well for larger ontologies, whereas FaCT++

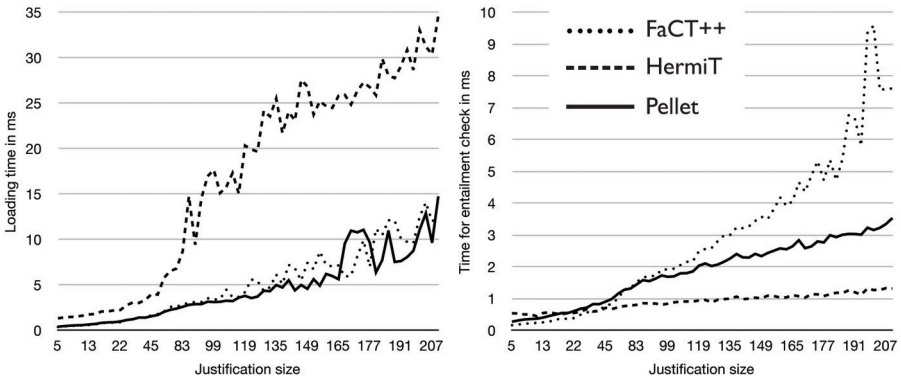

Fig. 3. Reasoner performance on an artificially generated ontology

suffers from quickly growing classification times. With respect to the lookup performance of FaCT++, is very likely that the JNI used in order to access the reasoner's native C++ code over the OWL API acts as a bottleneck that affects it negatively. The use of C++ code clearly affects the times for the calls to *dispose()*, as the FaCT++ framework has to perform actual memory management tasks in contrast to the two Java reasoners Pellet and HermiT which defer them indefinitely.

4.4 Application of JustBench

We propose our framework as a tool that helps reasoner developers as well as ontology engineers check the correctness and performance of reasoners on specific ontologies. With respect to ontology development, the framework allows ontology engineers to carry out fine-grained performance analysis of their ontologies and to *isolate* problems. While measuring the time for a full classification can give the developer information about the overall performance, it does not assist in finding out *why* the ontology is easy or hard for a particular reasoner. JustBench isolates minimal subsets of the ontology, which can then be analysed manually to find out which particular properties are hard for which reasoners. One strategy for mitigating performance problems is to introduce redundancy. Adding the entailments of particularly hard justifications to the ontology causes them to "mask" other, potentially harder, justifications for the entailment. This leads to the reasoner finding the easier justifications first, which may improve its performance when attempting to find out whether an entailment holds in an ontology.

Reasoner developers also benefit from the aforementioned level of detail of justification-based reasoner analysis. By restricting the analysis to small subsets, developers can detect reasoner weaknesses and trace the sources by inspecting

the respective justifications, which will help understanding and improving reasoner behaviour. Additionally, as shown in the previous section, the method also detects unsound reasoning which may not be exhibited otherwise.

5 Conclusion and Future Work

To our knowledge, JustBench is the first framework for analytic, realistic benchmarking of reasoning over OWL ontologies. Even though we have currently only examined a few ontologies, we find the general procedure of using meaningful subsets of real ontologies to be insightful and highly systematic. At the very least, it is a different way of interacting with an ontology and the reasoners.

Our current selection principle (i.e., selecting justifications) has proven fruitful: While justifications alone are not analytically complete (e.g., they fail to test non-entailment features), they score high on understandability and manipulability and can be related to overall ontology performance. Thus, arguably, justifications are a good "front line" kind of test for ontology developers.

Future work includes:

- **Improving the software:** While we believe we have achieved good independence from irrelevant system noise, we believe this can be refined further, which is critical given the typically small times we are working with. Furthermore, some OWL API functions (such as *prepareReasoner()*) do not have a tightly specified functionality. We will work with reasoner developers to ensure the telemetry functions we use are precisely described and comparable across reasoners.
- **Testing more ontologies:** We intend to examine a wide range of ontologies. Even our limited set revealed interesting phenomena. Working with substantively more ontologies will help refine our methodology and, we expect, support broader generalisations about ontology difficulty and reasoner performance.
- **More analytics:** Currently, we have been doing fairly crude correlations between "reasoner performance" and gross features of justifications (e.g., size). This can be considerably improved.
- **New selection principles:** As we have mentioned, modules are an obvious candidate, though there are significant challenges, not the least that the actual number of modules in real ontologies tends to be exponential in the size of the ontology [15]. Thus, we need a principle for determining and computing "interesting" modules. Other possible selection principles include "repaired" justifications and unions of justifications.

Furthermore, we intend to experiment with exposing users to our analysis methodology to see if this improves their experience of dealing with performance problems.

References

1. S. Bail, B. Parsia, and U. Sattler. The justificatory structure of OWL ontologies. In *OWLED*, 2010.
2. B. Boyer. Robust Java benchmarking. www.ibm.com/developerworks/java/library/j-benchmark1.html, 2008.
3. S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In *Proc. of KR-02*. Morgan Kaufmann Publishers, 2002.
4. B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular reuse of ontologies: Theory and practice. *J. of Artificial Intelligence Research*, 31:273–318, 2008.
5. B. Cuenca Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. OWL 2: The next step for OWL. *J. of Web Semantics*, 6(4):309–322, 2008.
6. D. J. Dewitt. The Wisconsin benchmark: Past, present, and future. In J. Gray, editor, *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.
7. J. V. Franco. On the probabilistic performance of algorithms for the satisfiability problem. *Inf. Process. Lett.*, 23(2):103–106, 1986.
8. T. Gardiner, D. Tsarkov, and I. Horrocks. Framework for an automated comparison of description logic reasoners. In *International Semantic Web Conference*, pages 654–667, 2006.
9. 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.
10. I. Horrocks and P. F. Patel-Schneider. Evaluating optimized decision procedures for propositional modal K(m) satisfiability. *J. Autom. Reasoning*, 28(2):173–204, 2002.
11. U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, editors, *KR*, pages 533–546. Morgan Kaufmann, 2002.
12. H. Lin and E. Sirin. Pellint - a performance lint tool for pellet. In C. Dolbear, A. Ruttenberg, and U. Sattler, editors, *OWLED*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
13. L. Ma, Y. Yang, Z. Qiu, G. T. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. In Y. Sure and J. Domingue, editors, *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2006.
14. Z. Pan. Benchmarking DL reasoners using realistic ontologies. In *OWLED*, 2005.
15. B. Parsia and T. Schneider. The modular structure of an ontology: An empirical study. In F. Lin, U. Sattler, and M. Truszczynski, editors, *KR*. AAAI Press, 2010.
16. S. Rudolph, T. Tserendorj, and P. Hitzler. What is approximate reasoning? In D. Calvanese and G. Lausen, editors, *RR*, volume 5341 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2008.
17. M. Schaerf and M. Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:249–310, 1995.
18. B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artif. Intell.*, 81(1-2):17–29, 1996.
19. T. Wang and B. Parsia. Ontology performance profiling and model examination: first steps. *Lecture Notes in Computer Science*, 4825:595, 2007.
20. T. Weithöner, T. Liebig, M. Luther, and S. Böhm. What's wrong with OWL benchmarks? In *SSWS*, 2006.