# The Serializable and Incremental Semantic Reasoner fuzzyDL

Ignacio Huitzil
*University of Zaragoza*
Zaragoza, Spain
ihuitzil@unizar.es

Umberto Straccia
*ISTI-CNR*
Pisa, Italy
straccia@isti.cnr.it

Carlos Bobed
*everis/NTT Data*
*Univ. Zaragoza*, Spain
cbobed@unizar.es

Eduardo Mena
*I3A, University of Zaragoza*
Zaragoza, Spain
emena@unizar.es

Fernando Bobillo
*I3A, University of Zaragoza*
Zaragoza, Spain
fbobillo@unizar.es

***Abstract*—Serializable and incremental semantic reasoners make it easier to reason on a mobile device with limited resources, as they allow the reuse of previous inferences computed by another device without starting from scratch. This paper describes an extension of the fuzzy ontology reasoner fuzzyDL to make it the first serializable and incremental semantic reasoner. We empirically show that the size of the serialized files is smaller than in another serializable semantic reasoner (JFact), and that there is a significant decrease in the reasoning time.**

***Index Terms*—fuzzy ontologies, semantic reasoning, incremental reasoning, mobile computing**

## I. Introduction

In the last years there is an increasing interest in the use of semantic reasoners in mobile devices [1], [2]. Semantic reasoners are software tools that can answer queries or compute inferences from logical knowledge bases represented using ontologies. An ontology is an explicit and formal specification of concepts, individuals, and relationships of the application domain at hand [3], [4]. Ontologies, encoded using the standard language OWL 2 [5], are nowadays widely used for knowledge representation favoring, among others, also information integration, the reuse of components, or discovering implicit knowledge.

We recall that, in the context of semantic reasoning on mobile devices, some reasoning strategies have been proposed so far: namely, local reasoning (all the reasoning is done on the device), external reasoning (the reasoning is done on an another device, such as a fast dedicated server, and sending the results to the mobile device), or hybrid reasoning (doing a part of the reasoning externally, and a part locally) [6]. Hybrid reasoning seems to be as a promising trade-off between the other options: on the one hand, it can benefit from the speed of an external device to preprocess the ontology (recall that mobile devices typically have limitations in terms of CPU power, memory, connectivity, etc.), and on the other hand it can also add new sensitive information without compromising it (the information is added in the user's device without disclosing it). Moreover, this approach does not require to communicate with the server too many times (typically, only to download the preprocessed ontology). This latter aspect is interesting because in mobile computing environments connectivity is often unreliable and additionally battery consuming.

I. Huitzil was partially funded by Universidad de Zaragoza - Santander Universidades (Ayudas de Movilidad para Latinoamericanos - Estudios de Doctorado). I. Huitzil, C. Bobed, E.Mena, and F. Bobillo were partially supported by the projects TIN2016-78011-C4-3-R and JIUZ-2018-TEC-02, and by DGA/FEDER.

To support hybrid reasoning, serializable incremental semantic reasoners are needed. Unfortunately, although there are some serializable and some incremental semantic reasoners, there are no semantic reasoners yet that are both serializable and incremental. The objective of this paper is to discuss an extension of the fuzzyDL reasoner [7] to fill this gap, making it the first serializable and incremental semantic reasoner.

While ontologies have proved to be very useful in many applications, in many real world domains knowledge is imprecise or vague. In such scenarios, fuzzy ontologies (fuzzy extensions of the ontologies with elements of fuzzy logic and fuzzy set theory [8], [9]) have been proposed [10]. The fuzzyDL reasoner is able to provide reasoning services over fuzzy ontologies and, because classical ontologies are a special case, also over classical ontologies. Managing imprecise information on mobile devices seems also very promising, but it has not been deeply studied (with some exceptions such as the fuzzy ontology-based recommender systems [11], [12]). Having a serializable and incremental fuzzy ontology reasoner might also be beneficial to increase the support for imprecise knowledge on mobile devices.

The remainder of the paper is organized as follows. Section II provides some background on serializable and incremental reasoners, and about the reasoner fuzzyDL. Section III describes the extension of the fuzzyDL reasoner to make it serializable and incremental. Next, Section IV reports an empirical evaluation of the size of the serialized files, the times to serialize and deserialize a reasoner, and the reasoning time. Finally, Section V compares our approach with related work and Section VI addresses conclusions and some ideas for future research.

## II. Background

### A. Serializable Incremental Reasoners

This section recalls main definitions and discussions in [6].

A *serializable* semantic reasoner can clone the data structures that represent its internal object state, obtaining two or more independent instances of the reasoner that can evolve in parallel. We will also assume that they can be written into a file. Typically, the file can be computed by a server and

978-1-7281-6932-3/20/$31.00 ©2020 IEEE

downloaded by a mobile device using hybrid reasoning. An example of serializable semantic reasoner is *JFact*.[1]

Serializable reasoners depend on the version of the reasoner: Small changes in the code of the reasoner could require changes in the serialization. They also require a common serialization strategy (e.g., a Java virtual machine –on the server– does not serialize data in the same way as a Dalvik/ART virtual machine –on an Android device).

A less restrictive concept is that of persistent semantic reasoner. A *persistent* semantic reasoner can save its internal state together with some precomputed inferences and reload it (for a given ontology). If it receives as input a previously considered ontology, it reuses previously computed calculations, avoiding the repetition of such calculations. For example, it can store the inferred class hierarchy obtained in a ontology classification.

On the other hand, a semantic reasoner is *incremental* if it can manage changes in the ontology without restarting the reasoning from scratch: that is, avoiding reloading the ontology and repeating computations (such as reclassifying the ontology). Incremental reasoners are useful, for example, when we want to submit several queries to the same ontology.

To implement hybrid reasoning on mobile devices, four strategies have been proposed:

1) The server can send an expanded ontology (with all the inferences explicitly represented) back.
2) If the mobile device has a copy of the ontology, the server can send only a list of the inferences and the axioms can be integrated on the mobile device.
3) If the reasoner is serializable, the external server can send instead a copy of the reasoner. The mobile device avoids the cost of loading and preprocessing the ontology, but requires that both devices (the server and the mobile) use the same reasoner and version.
4) If the mobile device has a copy of the ontology, the external server can provide a serialized version of the reasoner but not including the original ontology, which will be locally integrated. This requires some additional time to add the axioms but reduces the transmission size.

We will assume the third case. If the reasoner is incremental, we can add new axioms to it reusing the previous inferences.

### B. fuzzyDL reasoner

In this section we recap the main concepts and definitions in [7]. We assume the reader is familiar with Description Logics (DLs) [4] and the standard ontology language OWL 2 [5].

The fuzzy ontology reasoner fuzzyDL is publicly available.[2] It supports a very expressive language: a fuzzy extension of a fragment of the language OWL 2 extended with some unique features and capabilities of fuzzy logic. It supports different fuzzy logical operators (Łukasiewicz, Gödel, and Zadeh fuzzy logics, see Table I), but is backwards compatible and can also be used as a crisp ontology reasoner.

[1]http://jfact.sourceforge.net

[2]http://www.umbertostraccia.it/cs/software/fuzzyDL/fuzzyDL.html

| | **Gödel** | **Łukasiewicz** | **Zadeh** |
|---|---|---|---|
| Conjunction $\alpha \otimes \beta$ | $\min(\alpha, \beta)$ | $\max(\alpha + \beta - 1, 0)$ | $\min(\alpha, \beta)$ |
| Disjunction $\alpha \oplus \beta$ | $\max(\alpha, \beta)$ | $\min(\alpha + \beta, 1)$ | $\max(\alpha, \beta)$ |
| Implication $\alpha \Rightarrow \beta$ | $\begin{cases} 1 & \text{if } \alpha \leq \beta \\ \beta & \text{otherwise} \end{cases}$ | $\min(1 - \alpha + \beta, 1)$ | $\max\{1 - \alpha, \beta\}$ |
| Negation $\ominus \alpha$ | $\begin{cases} 1 & \text{if } \alpha = 0 \\ 0 & \text{otherwise} \end{cases}$ | $1 - \alpha$ | $1 - \alpha$ |

TABLE I
COMBINATION FUNCTIONS SUPPORTED BY FUZZYDL

The reasoning algorithm combines an extension of classical tableau algorithms with mathematical optimization. In particular, reasoning is reduced to a *Mixed Integer Linear Programming* (MILP) problem: the minimization/maximization of a $[0, 1]$-variable given a set of inequation constraints. In the classical semantics, all $[0, 1]$-variables are forced to take a value in $\{0, 1\}$. To solve the MILP problems, fuzzyDL uses Gurobi mathematical optimization solver.[3]

Regarding the language, it supports an extension of the fuzzy Description Logic $\mathcal{SHIF}(\mathbf{D})$. Supported OWL 2 concepts include atomic concepts, top concept, bottom concept, conjunction, disjunction, negation, and existential restrictions, universal restrictions, object-data value restrictions, and local reflexivity concepts. There are also some specific fuzzy concepts, such as the combination via aggregated operators.

Supported axioms include concept/role assertions, *General Concept Inclusions* (GCIs), role inclusions, role transitivity, role functionality, inverse roles, inverse role functionality, role symmetry, role reflexivity, and data property range axioms. GCIs include concept equivalences, concept disjointness, disjoint union of concepts, domain axioms, and object property range axioms. The main difference to the crisp variant is that now it is possible to state than an axiom holds to some degree of truth.

However, one of the main features of fuzzy ontologies are fuzzy datatypes, that make it possible to represent fuzzy sets using their membership function. Among others, fuzzyDL supports trapezoidal (Figure 1.a), triangular (Figure 1.b), left-shoulder (Figure 1.c), and right-shoulder (Figure 1.d) membership functions.

Regarding the traditional reasoning tasks, it supports ontology consistency, concept satisfiability, concept subsumption, entailment, and instance retrieval. There are also some tasks specific to the fuzzy case such as best entailment degree of an axiom, best satisfiability degree of a concept, variable maximization/minimization and defuzzification.

## III. SERIALIZABLE INCREMENTAL FUZZYDL

In this section we will give some details about the fuzzy ontology reasoner fuzzyDL. Firstly, we detail how we converted it into a serializable reasoner. Secondly, we discuss how turned it into an incremental reasoner.

[3]http://www.gurobi.com

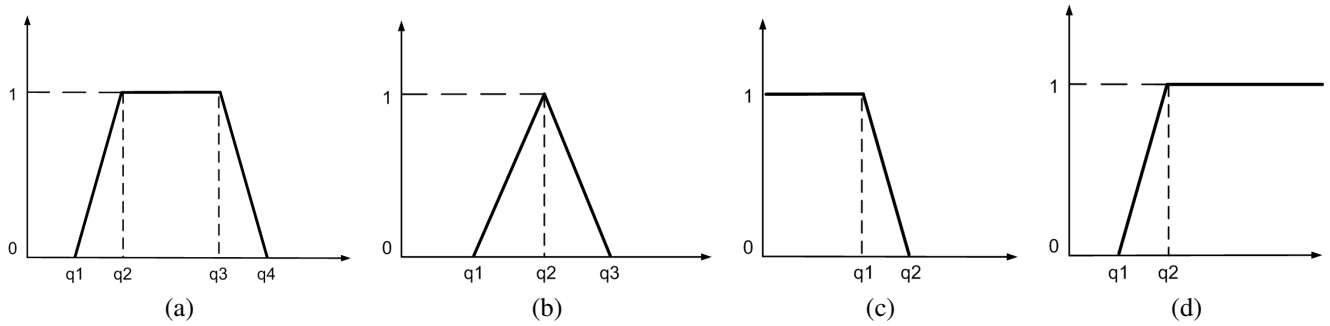


Fig. 1. (a) Trapezoidal; (b) Triangular; (c) Left-shoulder; (d) Right shoulder functions.

### A. Serializable fuzzyDL

In Java applications, to make a class serializable it has to implement an interface called Serializable. Furthermore, all other classes used by a serializable class must be serializable as well. This can be a problem if an application uses third-party libraries such that the source cannot be modified to implement the serializable class. Furthermore, serialization converts objects into bytes, but it does not affect class variable (static variables in Java).

fuzzyDL's main class KnowledgeBase encodes a reasoner state and a fuzzy ontology, not only with the original axioms but also with some inferred ones. In the serializable version there are two new methods:

- writeToFile makes it possible to save a KnowledgeBase object into a binary file.
- readFromFile obtains a KnowledgeBase object from a serialized binary file.

To make fuzzyDL serializable we needed to revise the code allowing to do three things:

- Ensure that all necessary classes (KnowledgeBase and the classes that it uses, e.g., class Individual) implement the Serializable interface.
- Encode class variables as object variables.
- Store the data using our own classes rather than Gurobi classes, acting as a wrapper. Thus, we have all the required data to create Gurobi objects when needed.

### B. Incremental fuzzyDL

fuzzyDL applies some preprocessing that transforms a given fuzzy ontology $\mathcal{O}$ into an expanded version that can be reused to answer different queries. For simplicity, we will describe here the preprocessing when behaving like a classical semantic reasoner (i.e., without managing fuzzy logic operators or degrees of truth), and it includes the following tasks (for more details, see [10], [7]):

1) Determine the language of the fuzzy ontology, e.g., $\mathcal{ALC}$. This is useful to know which inference optimizations methods can be applied.
2) Convert strings into integers. For each data property assertion of the form $T(i, s)$ where $s$ is a string, replace $s$ with an integer number. Integers are assigned in such a way that the lexicographic order of all strings in the ontology is preserved. This is needed in order to deal with string based operators within MILP.
3) Solve inverse roles. For each object property assertion of the form $R(i_1, i_2)$, it adds an assertion $R^I(i_1, i_2)$ if $R^I$ is an inverse role of $R$.
4) Compute the property hierarchy. For example, if $R_1$ is a sub-property if $R_2$ and $R_2$ is a sub-property of $R_3$, add that $R_1$ is a sub-property of $R_3$.
5) Solve object property assertions. For example, for each object property assertion of the form $R_1(i_1, i_2)$ and for each super property $R_2$ of $R_1$, we add an assertion $R(i_1, i_2)$. Furthermore, if there is a pair of assertions of the form $R_1(i_1, i_2)$ and $R_1(i_2, i_3)$ for a transitive role $R$, we add an assertion $R_1(i_1, i_3)$.
6) Solve reflexive roles. For each reflexive role $R$ and each individual $i$ in the ontology, add an assertion $R(i, i)$.
7) Solve functional roles. If there is a pair of assertions of the form $R_1(i_1, i_2)$ and $R_1(i_1, i_3)$ for a functional role $R$, then state that $i_2$ and $i_3$ must be the same individual.
8) Preprocess TBox. In the current version, there is an absorption algorithm that splits the TBox into an acyclic part and a general part [13]. In the acyclic part, it is possible to reason using an optimization called lazy unfolding. The intuitive idea is that TBox axioms are not applied to every individual but only to those individuals that are known to belong to some classes, decreasing the number of applications of the rules. In the general part, harder reasoning rules are needed and even simple TBox axioms (where the left side of the axiom is a named concept) are represented as GCIs. In the future, we plan to implement a classification algorithm to expand the class hierarchy (see [10] for a preliminary version).
9) Compute blocking type. Depending on the language of the fuzzy ontology, different blocking strategies are needed: subset (of labels), simple equality (of labels), simple pairwise, anywhere subset, anywhere equality, and anywhere pairwise [14]. Of course, one wants to use the simplest strategy that provides correct results for a given language.
10) Solve concept assertions. For each concept assertion $C(a)$ in the ontology, we apply some tableau rules to decompose $C$ into simpler concepts.

Once the preprocessing has been done, to solve a query, the reasoner reuses the expanded version of the fuzzy ontology, but creates a local copy. For instance, to check if an ontology entails a concept assertion of the form $C(a)$, fuzzyDL adds a new assertion of the form $(\neg C)(a)$. Since this new assertion is added to the local copy, it will not affect other future queries.

## IV. Evaluation

In this section, we address the evaluation of the extension of fuzzyDL described in this paper. We will firstly discuss the datasets and the setup. Then, we will describe two different types of experiments: a comparison of the size of the serialized ontologies computed by fuzzyDL and JFact, and an evaluation of the reasoning time after reusing already computed inferences. We only compare ourselves with JFact because it is the only serializable semantic reasoner.

### A. Experimental setup

To perform our evaluation we used various datasets:

- Firstly, we consider *Fuzzy Beer*, a fuzzy ontology with information about beers used in GimmeHop, a knowledge-based recommender system for mobile devices [12]. Fuzzy Beer ontology is able to represent concepts (e.g., beer types or breweries), object properties (e.g., between beers and breweries), data properties (e.g., alcohol level ABV, color, or bitterness), instances (e.g., beers and countries), and fuzzy datatypes. In particular, Fuzzy Beer has 15317 beer individuals.
- Secondly, we consider the 51 ontologies in the Absorption dataset developed in [13]. The idea of the dataset is to consider a fuzzy ontology (*Fuzzy Wine*) developed by humans, and other fuzzy ontologies randomly generated. For each crisp ontology, there are several fuzzy versions with different semantics and percentage of fuzzy axioms. In this paper, we will consider the original crisp ontologies and Fuzzy Wine.
- Finally, we consider the 36 large OWL 2 DL ontologies in the *ORE 2013* dataset [15]. ORE 2013 dataset contains 200 ontologies per profile (i.e., OWL 2 EL, OWL 2 RL, and OWL 2 DL) from the NCBO BioPortal[4], the Oxford Ontology Library[5], and the Manchester Ontology Repository[6]. Ontologies are classified according to their number of logical axioms as *small* ($\leq$ 500), *medium* (between 500 and 4999), and *large* ontologies ($\geq$ 5000).

We compare fuzzyDL with the crisp ontology reasoner JFact. Therefore, in this section fuzzyDL assumes a semantics based on classical logic. While fuzzyDL computes the preprocessing discussed in Section III-B, JFact uses the method precomputeInferences to compute the following axioms: class assertions, class hierarchy, object property assertions, data property assertions, object property hierarchy, data property hierarchy, same individuals, different individuals, and disjoint classes.

[4]http://bioportal.bioontology.org
[5]http://www.cs.ox.ac.uk/isg/ontologies
[6]http://rpc295.cs.man.ac.uk:8080/repository

During our experiments, we set a timeout of two hours to solve the required tasks. Experiments were repeated 5 times and we took the standard average of the computed values.

All experiments were performed on a laptop computer with Intel Core i7-8550U 1.8 GHz, 16 GB RAM under Windows 7 64-bits. The versions of the software were Java 1.8, JFact 4.0.4, OWL API 4.2.7, and Gurobi 8.1.0 build V8.1.0rc1 (Academic License); these are the versions of JFact and OWL API that were used in a comparison between JFact serialization and Fact++ persistence [6].

### B. Serialized Files: Size and Time

In this part of the evaluation, for each ontology in the datasets, we preprocess it, we serialize the reasoner (including the expanded fuzzy ontology) into a file, and we deserialize it. We compare fuzzyDL with JFact, the only serializable reasoner that we are aware of.

The results are shown in Table II for those ontologies that were successfully processed by both JFact anf fuzzyDL (datasets are separated using horizontal lines). For both reasoners we show three values:

- *SeriSize*: size in MB of the serialized reasoner (including the fuzzy ontology) after preprocessing the ontology.
- *SaveTime*: time in seconds needed to obtain a serialized version of the reasoner and to save it into a file.
- *LoadTime*: time in seconds needed to restore a version of the reasoner from a serialized file.

As we can see, fuzzyDL always computes smaller files. The differences are significant for (Fuzzy) Beer and for the Absorption dataset, but are quite impressive for OWL 2013. For example, while JFact requires 100.4 MB to serialize ontology 000004, fuzzyDL only uses 2 MB. It is worth to recall, however, that JFact does include some information that fuzzyDL does not (inferred class hierarchy).

Regarding the times, we can see that deserialization is slightly slower than serialization. As both reasoners use the Java serialization strategy, and because fuzzyDL manages smaller files, it is not surprising that fuzzyDL is always faster. The differences can also be very important; for the example discussed in the previous paragraph, fuzzyDL requires 0.64 s to serialize and 1.28 s to restore, whereas JFact requires 62.78 s and 86.26 s, respectively.

It is also worth noting that there were 66 ontologies (75 %) where at least one of the two reasoners failed, 31 in the Absorption dataset (61 % of the dataset) and 35 in the ORE 2013 dataset (97 % of the dataset). In particular, JFact failed in 53 and fuzzyDL in 40. Focusing on fuzzyDL, we found the following problems:

- 21 timeouts,
- 12 ontologies included OWL 2 elements that are not currently supported by fuzzyDL, e.g., object property chains, cardinality restrictions, enumerations, or universal data property restrictions.
- 2 parsing errors when importing the ontologies (for example, because of a non-ASCII character "á"), and
- 5 null pointer exceptions, requiring further investigation.

| | JFact | | | FuzzyDL | | |
|---|---|---|---|---|---|---|
| Ontology | SeriSize (MB) | SaveTime (s) | LoadTime (s) | SeriSize (MB) | SaveTime (s) | LoadTime (s) |
| Beer | 120.89 | 65.96 | 91.88 | 18.95 | 7.46 | 11.47 |
| amino-acid | 0.22 | 0.25 | 0.30 | 0.04 | 0.03 | 0.04 |
| cancer_my | 0.31 | 0.32 | 0.40 | 0.04 | 0.04 | 0.05 |
| chemical | 0.16 | 0.22 | 0.27 | 0.02 | 0.03 | 0.02 |
| EMAP.obo | 26.41 | 8.67 | 12.72 | 2.85 | 1.34 | 2.14 |
| FMA | 1079.97 | 104.26 | 150.88 | 43.71 | 25.13 | 34.71 |
| FuzzyWine | 1.24 | 0.88 | 1.17 | 0.17 | 0.17 | 0.23 |
| galen-ians-full-doctored | 6.67 | 3.57 | 5.5 | 1.35 | 0.42 | 0.74 |
| gene_ontology_edit.obo | 6.67 | 3.54 | 5.19 | 4.97 | 2.59 | 3.83 |
| goslim | 0.30 | 0.18 | 0.27 | 0.05 | 0.06 | 0.06 |
| lubm | 8.66 | 4.76 | 6.86 | 3.83 | 3.02 | 3.90 |
| matchmaking | 0.27 | 0.30 | 0.37 | 0.03 | 0.03 | 0.03 |
| pathway.obo | 0.82 | 0.54 | 0.72 | 0.09 | 0.07 | 0.09 |
| people.fd | 0.25 | 0.53 | 0.65 | 0.04 | 0.04 | 0.05 |
| pizza | 0.38 | 0.40 | 0.45 | 0.05 | 0.08 | 0.12 |
| po | 0.86 | 0.56 | 0.74 | 0.06 | 0.08 | 0.08 |
| SIGKDD-EKAW | 0.34 | 0.33 | 0.42 | 0.03 | 0.03 | 0.04 |
| so-xp.obo | 2.64 | 1.39 | 2.01 | 0.31 | 0.17 | 0.24 |
| spatial.obo | 0.26 | 0.28 | 0.36 | 0.05 | 0.03 | 0.05 |
| teleost_taxonomy.obo | 33.58 | 19.37 | 27.90 | 4.84 | 2.63 | 3.81 |
| worm_phenotype_xp.obo | 2.90 | 1.38 | 2.20 | 0.46 | 0.18 | 0.30 |
| teleost-taxonomy.1081 | 27.60 | 23.11 | 27.83 | 5.20 | 3.22 | 5.31 |

TABLE II
SERIALIZATION OF THE FUZZY BEER AND ABSORPTION ONTOLOGIES USING *JFact* AND *FuzzyDL*.

### C. Evaluation of the Reasoning Time

In this part of the evaluation, we focus on the reasoning time of the serialized incremental version of fuzzyDL. We do not compare fuzzyDL with JFact because it is not incremental (see Section V for a discussion). For each of the 48 ontologies where fuzzyDL did not fail, Table III shows several values:

- *LoadTime*: time to load the ontology from a text file. This value is used to compare with the non-incremental version of the reasoner.
- *CloneTime*: time to obtain a copy of an object representing the reasoner. This value is used to compare with the time needed to load the state of a reasoner from its serialization.
- *PreprocessTime*: time to preprocess the ontology, computing all the inferences that can be shared when solving any query.
- *SubTime*: time to solve a concept subsumption query, assuming that the ontology has already been preprocessed. This query considers two randomly selected named concepts in the ontology.
- *SatTime*: time to solve a concept satisfiability query, assuming that the ontology has already been preprocessed. This query considers a randomly selected named concept in the ontology.
- *EntTime*: time to solve an entailment query, assuming that the ontology has already been preprocessed. In particular, we consider the entailment of a concept assertion axiom involving an individual and a named concept in the ontology, both randomly selected.

Based on these values, Table IV shows the following times:[7]

- *TimeR* (restore): Time to prepare incremental and serializable reasoning, i.e., time to deserialize the reasoner.
- *TimeO* (old): Time to solve a query without incremental reasoning. This includes the time to load the ontology from a text file, the time to preprocess it, and the time to solve the query.
- *TimeD* (download): Time to obtain a remote serialized file and to prepare incremental and serializable reasoning. This includes the time to download the file, and the time to deserialize the reasoner. The time to download the file is estimated by dividing the size file (shown in Table II) by the data transfer speed. The data transfer speed depends on the technology (e.g., WiFi, mobile broadband, ...) and is typically rather variable; we have assumed 17.6 Mbps, as it was the global (after analyzing 87 countries) average mobile connection in 2019 [16].
- *TimeQ* (query): Time to solve the queries using incremental and serializable reasoning: time to clone the restored version of the reasoner plus time to solve the query.

Being incremental clearly decreases the reasoning time: to answer the first query, the reasoning time is the same (because both need to expand the ontology and solve the query), but for the next queries, it is possible to save the PreprocessTime. Instead, one need to compute a local copy of the reasoner (CloneTime), which is much faster. We can indeed see that TimeQ is always smaller than TimeO, for all query types. The decrease in the reasoning time is modest for easy ontologies, but can be quite significant for relatively complex ones. For example, in Fuzzy Beer, the incremental version requires 0.06s instead of 12.89s. However, there are also ontologies where PreprocessTime is much smaller than the query time (e.g., subsumption in Pizza), so in this case the decrease of the reasoning time in the incremental version is very small.

[7]Note that, while TimeR and TimeD are independent of the query type, TimeO and TimeQ have to be considered for each query type (concept subsumption, concept satisfiability, and entailment of a concept assertion).

| Ontology | LoadTime (s) | CloneTime (s) | PreprocessTime (s) | SubTime (s) | SatTime (s) | EntTime (s) |
|---|---|---|---|---|---|---|
| Beer | 0.95 | 0.040 | 11.91 | 0.02 | 2247.33 | 2250.22 |
| amino-acid | 0.02 | 0.0004 | 0.003 | 0.87 | 0.27 | Timeout |
| atom-common | 0.01 | 0.0004 | 0.001 | 0.02 | 0.008 | Timeout |
| cancer_my | 0.01 | 0.001 | 0.003 | 0.11 | 2.75 | 2.65 |
| chebi | 3.36 | 1.823 | 0.55 | 0.09 | 0.01 | Timeout |
| chemical | 0.01 | 0.0004 | 0.002 | 5.00 | 4.59 | Timeout |
| cton | 0.17 | 0.005 | 0.10 | 0.03 | 0.007 | Timeout |
| earthrealm | 0.05 | 0.003 | 0.02 | 0.39 | 0.36 | 0.34 |
| Economy | 0.01 | 0.003 | 0.02 | 0.01 | 0.07 | 0.06 |
| EMAP.obo | 0.11 | 0.005 | 0.05 | 0.02 | 0.006 | Timeout |
| FuzzyWine | 0.09 | 0.001 | 0.02 | 0.02 | 284.51 | 272.19 |
| fmaOwlDlComponent_1_4_0 | 0.44 | 0.007 | 0.09 | 0.07 | Timeout | Timeout |
| FMA | 0.96 | 0.167 | 0.33 | 0.12 | 0.029 | Timeout |
| galen-ians-full-doctored | 0.08 | 0.007 | 0.04 | Timeout | 0.008 | Timeout |
| gene_ontology_edit.obo | 0.21 | 0.012 | 0.09 | 0.02 | 0.009 | Timeout |
| goslim | 0.01 | 0.001 | 0.001 | 0.004 | 0.02 | 0.02 |
| lubm | 0.38 | 0.007 | 2.31 | 0.003 | 495.16 | 516.7 |
| matchmaking | 0.01 | 0.0004 | 0.001 | 0.03 | 0.002 | Timeout |
| mygrid-moby-service | 0.04 | 0.0006 | 0.01 | 0.02 | 0.002 | Timeout |
| NCI | 0.25 | 0.011 | 0.13 | 0.04 | 0.02 | Timeout |
| pathway.obo | 0.02 | 0.0004 | 0.006 | 0.01 | 0.002 | Timeout |
| people.fd | 0.01 | 0.001 | 0.002 | 0.01 | 0.85 | 0.85 |
| periodic-table-complex | 0.01 | 0.0002 | 0.004 | 0.07 | 0.03 | Timeout |
| pizza | 0.04 | 0.000 | 0.01 | 163.51 | 0.01 | 0.01 |
| po | 0.01 | 0.001 | 0.003 | 0.81 | 0.84 | 0.81 |
| process | 0.07 | 0.003 | 0.02 | 0.44 | 0.37 | 0.40 |
| propreo | 0.03 | 0.001 | 0.01 | Timeout | 394.95 | Timeout |
| relative-places | 0.01 | 0.0002 | 0.001 | 0.10 | 0.06 | Timeout |
| SIGKDD-EKAW | 0.02 | 0.0004 | 0.002 | 0.41 | 0.002 | Timeout |
| so-xp.obo | 0.04 | 0.001 | 0.02 | 0.02 | 0.002 | Timeout |
| spatial.obo | 0.02 | 0.0002 | 0.006 | 0.06 | 0.002 | Timeout |
| subatomic-particle-complex | 0.02 | 0.0004 | 0.003 | 0.10 | 0.06 | Timeout |
| teleost_taxonomy.obo | 0.17 | 0.017 | 0.08 | 0.03 | 0.01 | Timeout |
| thesaurus | 0.67 | 0.031 | 0.37 | 0.40 | 0.19 | 0.20 |
| Transportation | 0.01 | 0.001 | 0.004 | 0.01 | 0.03 | 0.03 |
| worm_phenotype_xp.obo | 0.05 | 0.002 | 0.01 | 0.01 | 0.003 | Timeout |
| 00035 | 0.13 | 0.003 | 0.11 | Timeout | 0.01 | Timeout |
| 00368 | 0.40 | 0.216 | 0.15 | 0.02 | 0.01 | 305.19 |
| 00371 | 0.48 | 0.167 | 0.30 | Timeout | 0.01 | 346.26 |
| 00374 | 0.66 | 0.166 | 0.43 | 0.03 | 0.01 | 388.31 |
| 00386 | 0.50 | 0.192 | 0.30 | Timeout | 0.01 | 343.11 |
| 00390 | 0.41 | 0.170 | 0.29 | Timeout | 0.01 | 304.40 |
| 00398 | 0.46 | 0.165 | 0.28 | Timeout | 0.01 | 346.48 |
| 00400 | 0.43 | 0.163 | 0.36 | Timeout | 0.01 | 421.10 |
| 290113a0-5a1b-4f85-a716-ced96a6499e9__links | 0.22 | 0.013 | 0.10 | 0.01 | Timeout | 5.15 |
| d0e20d33-6bfa-4115-aba4-3a3f4ba8d586_mplied | 0.21 | 0.007 | 0.16 | 0.02 | 0.01 | Timeout |
| d5c7f91d-b5eb-4af1-9293-d90e7ff63b1e_1070 | 0.30 | 0.011 | 0.14 | 0.06 | 0.01 | Timeout |
| teleost-taxonomy.1081 | 0.26 | 0.021 | 0.13 | 0.05 | 0.02 | Timeout |

TABLE III
EVALUATION OF DIFFERENT PARTS OF THE REASONING IN FUZZYDL.

The time to restore the ontology (TimeR) can be significant, even higher than TimeO, but the advantage is that only needs to be computed once for a given ontology. If we also need to download the serialized version of the reasoner (TimeD), the time slightly increases but because it is only done once, the decrease in the reasoning time makes it worth.

## V. RELATED WORK

There are several semantic reasoners that are serializable or incremental but none of them is serializable and incremental.

- *JFact* is serializable in the versions 3.5.* and 4.0.*. It also takes advantage of the Java mechanisms for serialization and is able to save a binary file. However, incremental reasoning is not implemented in those versions, so if one adds new axioms it is necessary to start from scratch[8].
- *FaCT++*n [17][9] is claimed to be incremental (only in the non-buffered mode[10]) and persistent, although not serializable. Indeed, it is able to save a text file with a representation of the ontology (with some changes, e.g., URIs are encoded as integers), the reasoner state, etc. Being persistent could be acceptable sometimes, but we have checked that incremental reasoning using a restored version of the reasoner over a serialized ontology does not always give the correct results.

[8]Personal communication with Ignazio Palmisano (current main developer).
[9]http://owl.man.ac.uk/factplusplus
[10]Personal communication Dmitry Tsarkov (current main developer).

| Ontology | TimeR | TimeD (s) | Subsumption | | Satisfiability | | Entailment | |
|---|---|---|---|---|---|---|---|---|
| | | | TimeO (s) | TimeQ (s) | TimeO (s) | TimeQ (s) | TimeO (s) | TimeQ (s) |
| Beer | 11.47 | 20.08 | 12.89 | 0.06 | 2260.19 | 2247.37 | 2263.09 | 2250.26 |
| amino-acid | 0.04 | 0.06 | 0.89 | 0.87 | 0.28 | 0.27 | Timeout | Timeout |
| atom-common | 0.02 | 0.02 | 0.03 | 0.02 | 0.02 | 0.01 | Timeout | Timeout |
| cancer_my | 0.05 | 0.07 | 0.12 | 0.11 | 2.76 | 2.76 | 2.66 | 2.65 |
| chebi | 157.46 | 223.07 | 4.00 | 1.91 | 3.92 | 1.83 | Timeout | Timeout |
| chemical | 0.02 | 0.03 | 5.01 | 5.00 | 4.60 | 4.59 | Timeout | Timeout |
| cton | 2.15 | 3.51 | 0.31 | 0.04 | 0.28 | 0.01 | Timeout | Timeout |
| earthrealm | 0.80 | 1.11 | 0.45 | 0.39 | 0.43 | 0.36 | 0.40 | 0.34 |
| EMAP.obo | 2.14 | 3.44 | 0.18 | 0.02 | 0.17 | 0.01 | Timeout | Timeout |
| Economy | 0.47 | 0.72 | 0.03 | 0.01 | 0.09 | 0.07 | 0.09 | 0.07 |
| FuzzyWine | 0.23 | 0.30 | 0.13 | 0.02 | 284.62 | 284.52 | 272.3 | 272.19 |
| fmaOwlDlComponent_1_4_0 | 2.26 | 3.45 | 0.60 | 0.08 | Timeout | Timeout | Timeout | Timeout |
| FMA | 34.71 | 54.58 | 1.41 | 0.29 | 1.31 | 0.20 | Timeout | Timeout |
| galen-ians-full-doctored | 0.74 | 1.36 | Timeout | Timeout | 0.13 | 0.01 | Timeout | Timeout |
| gene_ontology_edit.obo | 3.83 | 6.09 | 0.33 | 0.04 | 0.31 | 0.02 | Timeout | Timeout |
| goslim | 0.06 | 0.09 | 0.01 | 0.01 | 0.03 | 0.02 | 0.03 | 0.02 |
| lubm | 3.90 | 5.64 | 2.69 | 0.01 | 497.85 | 495.17 | 519.39 | 516.71 |
| matchmaking | 0.03 | 0.05 | 0.04 | 0.03 | 0.02 | 0.003 | Timeout | Timeout |
| mygrid-moby-service | 0.12 | 0.19 | 0.08 | 0.02 | 0.05 | 0.003 | Timeout | Timeout |
| NCI | 4.36 | 7.73 | 0.43 | 0.05 | 0.40 | 0.03 | Timeout | Timeout |
| pathway.obo | 0.09 | 0.13 | 0.03 | 0.01 | 0.03 | 0.003 | Timeout | Timeout |
| people.fd | 0.05 | 0.07 | 0.02 | 0.02 | 0.85 | 0.85 | 0.85 | 0.85 |
| periodic-table-complex | 0.05 | 0.07 | 0.08 | 0.07 | 0.05 | 0.03 | Timeout | Timeout |
| pizza | 0.12 | 0.15 | 163.56 | 163.51 | 0.06 | 0.01 | 0.06 | 0.01 |
| po | 0.08 | 0.11 | 0.82 | 0.81 | 0.85 | 0.84 | 0.82 | 0.81 |
| process | 0.98 | 1.30 | 0.53 | 0.45 | 0.46 | 0.37 | 0.49 | 0.40 |
| propreo | 0.10 | 0.16 | Timeout | Timeout | 394.99 | 394.95 | Timeout | Timeout |
| relative-places | 0.02 | 0.03 | 0.11 | 0.10 | 0.08 | 0.06 | Timeout | Timeout |
| SIGKDD-EKAW | 0.04 | 0.05 | 0.42 | 0.41 | 0.02 | 0.003 | Timeout | Timeout |
| so-xp.obo | 0.24 | 0.38 | 0.08 | 0.02 | 0.06 | 0.002 | Timeout | Timeout |
| spatial.obo | 0.05 | 0.07 | 0.08 | 0.06 | 0.02 | 0.003 | Timeout | Timeout |
| subatomic-particle-complex | 0.05 | 0.07 | 0.12 | 0.10 | 0.08 | 0.06 | Timeout | Timeout |
| teleost_taxonomy.obo | 3.81 | 6.01 | 0.28 | 0.04 | 0.27 | 0.02 | Timeout | Timeout |
| thesaurus | 11.47 | 21.04 | 1.44 | 0.43 | 1.23 | 0.22 | 1.24 | 0.23 |
| Transportation | 0.21 | 0.28 | 0.02 | 0.01 | 0.05 | 0.03 | 0.04 | 0.03 |
| worm_phenotype_xp.obo | 0.30 | 0.51 | 0.07 | 0.01 | 0.06 | 0.01 | Timeout | Timeout |
| 00035 | 2.09 | 3.98 | Timeout | Timeout | 0.25 | 0.01 | Timeout | Timeout |
| 00368 | 21.36 | 33.84 | 0.57 | 0.24 | 0.55 | 0.22 | 305.73 | 305.40 |
| 00371 | 20.87 | 33.39 | Timeout | Timeout | 0.78 | 0.18 | 347.03 | 346.42 |
| 00374 | 28.93 | 41.45 | 1.12 | 0.20 | 1.10 | 0.17 | 389.40 | 388.47 |
| 00386 | 27.02 | 39.56 | Timeout | Timeout | 0.80 | 0.20 | 343.90 | 343.30 |
| 00390 | 20.7 | 32.53 | Timeout | Timeout | 0.71 | 0.18 | 305.10 | 304.57 |
| 00398 | 21.06 | 33.25 | Timeout | Timeout | 0.75 | 0.17 | 347.22 | 346.65 |
| 00400 | 22.59 | 35.61 | Timeout | Timeout | 0.81 | 0.17 | 421.89 | 421.26 |
| 290113a0-5a1b-4f85-a716-ced96a6499e9__links | 3.99 | 6.33 | 0.35 | 0.04 | 0.33 | 0.02 | Timeout | Timeout |
| d0e20d33-6bfa-4115-aba4-3a3f4ba8d586_mplied | 3.06 | 4.95 | 0.39 | 0.03 | 0.38 | 0.01 | Timeout | Timeout |
| d5c7f91d-b5eb-4af1-9293-d90e7ff63b1e_1070 | 6.10 | 7.99 | 0.51 | 0.07 | 0.46 | 0.02 | Timeout | Timeout |
| teleost-taxonomy.1081 | 5.46 | 7.82 | 0.45 | 0.07 | 0.41 | 0.04 | Timeout | Timeout |

TABLE IV
REASONING TIMES OF THE CLASSICAL VERSION AND THE SERIALIZABLE VERSION OF FUZZYDL.

- *Pellet* is incremental and persistent [18].[11] As in Fact++, it uses Java serialization to save a binary file with the reasoner state. It is also worth to remark that in Pellet 2.2 the incremental version of the reasoner does not support datatypes [19], and the situation seems similar in the most recent version 2.3. Unfortunately, datatypes are crucial in mobile and dynamic scenarios, as well as in fuzzy ontologies (see, e.g. [10], [20]).
- Finally, other semantic reasoners, such as *CEL* [21][12], *ELK* [22][13], and *SnoRocket* [23][14], implement some kind of incremental reasoning but, to the best of our knowledge, do not support serialization.

It is also worth to note that the support for incremental reasoning is usually restricted to the non-buffered mode. In the buffered mode, ontology changes are stored in a buffer and are only taken into account when the user invokes a flushing method. In the non-buffered mode, ontology changes are processed as soon as they are received. Currently, fuzzyDL does not implement a buffered mode.

The advantages of having semantic reasoners that are serializable or incremental has previously discussed in [6]. Another possible application is the implementation of a semantic reasoner managing volatile information. The idea was not to develop a new reasoner from scratch, but to build a *metarea-*

[11] http://clarkparsia.com/pellet
[12] https://tu-dresden.de/ing/informatik/thi/lat/forschung/software/cel
[13] http://liveontologies.github.io/elk-reasoner
[14] http://github.com/aehrc/snorocket

*soner* using a serializable and incremental semantic reasoner. In particular, the authors discuss a Java implementation using the *OWL API* [24][15] that would be able to use any serializable and incremental ontology reasoner accessible via the OWL API. Currently, fuzzyDL does not implement the OWL API.

## VI. Conclusions and future work

In this paper we have presented a new version of the fuzzy ontology reasoner fuzzyDL to make it the first semantic reasoner that is both serializable and incremental. Such features are particularly interesting for devices with limited resources, such as some mobile devices.

fuzzyDL can expand a fuzzy ontology with some inferences that can be reused when answering different queries. Moreover, it is possible to serialize the Java object that represents the reasoner and save it into a file. Classical ontologies are a special case of fuzzy ontologies, and fuzzyDL supports a very important fragment of the standard ontology language OWL 2.

Our experiments show that fuzzyDL computes smaller serialized files than JFact, the only other semantic reasoner that is serializable. fuzzyDL is also faster at both serializing and deserializing. While being incremental is helpful at decreasing the reasoning time, being also serializable slightly increases the cost of the first query because it is necessary to restore the serialized version of the file. We have also estimated the cost of downloading the file from a remote server before restoring the reasoner and it seems acceptable. Therefore, the idea of reusing from a mobile device a fuzzy ontology that was previously expanded in a different place (e.g., in a fast dedicated server) seems promising.

Future work might include developing a version of fuzzyDL working on mobile devices. Because it is implemented in Java, it seems easier to develop an Android version. So far, the only problem is that it uses a third-party library (Gurobi) for which currently there is no Android version. A possibility could be to replace Gurobi for another library solving mathematical optimization problems (in particular, Mixed Integer Linear Programming problems) completely developed in the fragment of Java compatible with Android.

With this Android version, one could investigate whether in mobile devices with limited resources the time to expand a fuzzy ontology, which is expected to be higher, will be higher than the deserialization time more often than in our evaluation.

Another obvious idea is the development of a new version of fuzzyDL supporting the OWL API, so that it is possible to manage volatile information as proposed in [6]. Fortunately, to make the communication between the metareasoner and fuzzyDL possible, it might be possible to implement only a very small fragment of the OWL API.

Last but not least, fuzzyDL parser to load OWL 2 ontologies could be improved (we found some bugs in ontologies encoded in the fragment of OWL 2 that fuzzyDL supports) and fuzzyDL preprocessing could be extended (e.g., with class classification [10]) in order to reduce the query time.

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

## References


[1] R. Yus and P. Pappachan, "Are apps going semantic? A systematic review of semantic mobile applications," in *Proceedings of the 1st International Workshop on Mobile Deployment of Semantic Technologies (MoDeST 2015)*, CEUR Workshop Proceedings 1506, 2015, pp. 2–13.

[2] C. Bobed, R. Yus, F. Bobillo, and E. Mena, "Semantic reasoning on mobile devices: Do androids dream of efficient reasoners?" *Journal of Web Semantics*, vol. 35, no. 4, pp. 167–183, 2015.

[3] S. Staab and R. Studer, Eds., *Handbook on Ontologies*, ser. International Handbooks on Information Systems, 2004.

[4] M. Krötzsch, F. Simančík, and I. Horrocks, "Description logics," *IEEE Intelligent Systems*, vol. 29, no. 1, pp. 12–19, 2014.

[5] B. Cuenca-Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler, "OWL 2: The next step for OWL," *Journal of Web Semantics*, vol. 6, no. 4, pp. 309–322, 2008.

[6] C. Bobed, F. Bobillo, E. Mena, and J. Z. Pan, "On serializable incremental semantic reasoners," in *Proceedings of the 9th International Conference on Knowledge Capture (K-CAP 2017)*. ACM, December 2017, pp. 187–190.

[7] F. Bobillo and U. Straccia, "The fuzzy ontology reasoner fuzzyDL," *Knowledge-Based Systems*, vol. 95, pp. 12–34, 2016.

[8] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[9] G. J. Klir and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications*. Prentice-Hall, Inc., 1995.

[10] U. Straccia, *Foundations of Fuzzy Logic and Semantic Web Languages*, ser. CRC Studies in Informatics Series. Chapman & Hall, 2013.

[11] J. A. Morente-Molinera, R. Wikström, E. Herrera-Viedma, and C. Carlsson, "A linguistic mobile decision support system based on fuzzy ontology to facilitate knowledge mobilization," *Decision Support Systems*, vol. 81, pp. 66–75, 2016.

[12] I. Huitzil, F. Alegre, and F. Bobillo, "GimmeHop: A recommender system for mobile devices using ontology reasoners and fuzzy logic," *Fuzzy Sets and Systems*, 2020.

[13] F. Bobillo and U. Straccia, "Optimising fuzzy description logic reasoners with general concept inclusions absorption," *Fuzzy Sets and Systems*, vol. 292, pp. 98–129, 2016.

[14] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: An OWL 2 reasoner," *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245–269, 2014.

[15] R. S. Gonçalves, S. Bail, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia, B. Glimm, and Y. Kazakov, "OWL Reasoner Evaluation (ORE) workshop 2013 results: Short report," in *Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE 2013)*. CEUR Workshop Proceedings 1015, 2013, pp. 1–18.

[16] P. Boyland, "The state of mobile network experience - Benchmarking mobile on the eve of the 5G revolution," 2019, http://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2019-05/the_state_of_mobile_experience_may_2019_0.pdf.

[17] D. Tsarkov, "Incremental and persistent reasoning in FaCT++," in *Proceedings of the 3rd Internat. Workshop on OWL Reasoner Evaluation (ORE 2014)*. CEUR Workshop Proceedings 1207, 2014, pp. 16–22.

[18] E. Sirin, B. Parsia, B. Cuenca-Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Journal of Web Semantics*, vol. 5, no. 2, pp. 51–53, 2007.

[19] C. Bobed, F. Bobillo, S. Ilarri, and E. Mena, "Answering continuous description logic queries: Managing static and volatile knowledge in ontologies," *International Journal on Semantic Web and Information Systems*, vol. 10, no. 3, pp. 1–44, 2014.

[20] F. Bobillo and U. Straccia, "Fuzzy ontology representation using OWL 2," *International Journal of Approximate Reasoning*, vol. 52, pp. 1073–1094, 2011.

[21] F. Baader, C. Lutz, and B. Suntisrivaraporn, "CEL – A polynomial-time reasoner for life science ontologies," in *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR 2006)*, Lecture Notes in Artificial Intelligence 4130,2006, pp. 287–291.

[22] Y. Kazakov, M. Krötzsch, and F. Simančík, "The incredible ELK," *Journal of Automated Reasoning*, vol. 53, pp. 1–61, 2014.

[23] M. Lawley and C. Bousquet, "Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner," in *Proceedings of the Australasian Ontology Workshop 2010 (AOW 2010)*, 2010, pp. 45–50.

[24] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies," *Semantic Web Journal*, vol. 2, no. 1, pp. 11–21, 2011.