Just: a Tool for Computing Justifications w.r.t. $\mathcal{ELH}$ Ontologies

Michel Ludwig*

Theoretical Computer Science, TU Dresden, Germany
michel@tcs.inf.tu-dresden.de

Abstract. We introduce the tool JUST for computing justifications for general concept inclusions w.r.t. ontologies formulated in the description logic $\mathcal{EL}$ extended with role inclusions. The computation of justifications in JUST is based on saturating the input axioms under all possible inferences w.r.t. a consequence-based calculus. We give an overview of the implemented techniques and we conclude with an experimental evaluation of the performance of JUST when applied on several practical ontologies.

1 Introduction

Users or authors of ontologies are often interested in understanding why a consequence α logically follows from a given ontology $\mathcal{T}$. As the ontologies that are typically used in practice consist of several thousand axioms, it can be helpful to compute a small subset of $\mathcal{T}$ which entails the consequence α for analysing why α logically follows from $\mathcal{T}$. A minimal such subset is called a *justification* of α w.r.t. $\mathcal{T}$. Furthermore, when α represents an erroneous consequence that should be fixed, it becomes necessary to compute *all* the justifications of α w.r.t. $\mathcal{T}$ to ensure that all the relevant axioms of $\mathcal{T}$ are taken into account when changing the ontology in such a way that α is no longer entailed.

Previous approaches described in the literature for computing all the justifications of an ontology w.r.t. a consequence can broadly be classified into the following two categories. "Black-box" algorithms use off-the-shelf reasoners solely to check whether a concept inclusion follows from an ontology while searching for candidate subsets of the ontology that can serve as justifications [4,8,9]. "Glass-box" approaches, on the other hand, typically rely on intricate modifications to a pre-existing reasoner implementation to keep track of which axioms were used during the classification of an ontology. This information can then be used to construct a justification [2,5,11]. In this paper we follow such a "glass-box" approach as well. Generally speaking, the transformation of a reasoner into a tool capable of computing all justifications is not an easy task as one goal of efficient reasoners is to avoid generating duplicate inferences for the same consequence.

We present the tool JUST that is designed from the ground up for computing either one, or all the justifications for general concept inclusions w.r.t. ontologies formulated in the description logic $\mathcal{EL}$ extended with role inclusions. We

* The author was supported by the German Research Foundation (DFG) within the Cluster of Excellence 'Center for Advancing Electronics Dresden' (cfAED).

first give a brief overview of the implemented techniques, and we conclude the paper with an experimental evaluation of its performance on several practical ontologies. The tool and proofs establishing its correctness and completeness are available from `http://lat.inf.tu-dresden.de/~michel/software/just/`

2 Preliminaries

We assume the reader to be familiar with basic notions of description logics [1]. An $\mathcal{ELH}$-ontology, or $\mathcal{ELH}$-TBox $\mathcal{T}$, is a finite set of axioms of the form $C \sqsubseteq D$ or $r \sqsubseteq s$, where C, D are $\mathcal{EL}$-concepts and r, s are roles names. An $\mathcal{EL}$-concept C is either a concept name, or a concept that makes use of the concept constructors $\top$, $\sqcap$, $\exists r.D$ only, where r is a role name and D an $\mathcal{EL}$-concept. We assume standard set-theoretic semantics.

We now formally introduce the main notion relevant for our tool.

Definition 1. *Let $\mathcal{T}$ be an $\mathcal{ELH}$-TBox and let C, D be $\mathcal{EL}$-concepts. A* justification[1] *for $\mathcal{T} \models C \sqsubseteq D$ is a subset $\mathcal{M} \subseteq \mathcal{T}$ such that $\mathcal{M} \models C \sqsubseteq D$, and for every $\mathcal{M}' \subsetneq \mathcal{M}$ it holds that $\mathcal{M}' \not\models C \sqsubseteq D$. The set of all the justifications for $\mathcal{T} \models C \sqsubseteq D$ will be denoted by $\mathsf{Just}_{\mathcal{T}}(C \sqsubseteq D)$.*

Example 1. Let $\mathcal{T} = \{A \sqsubseteq X,\ A \sqsubseteq Y,\ X \sqsubseteq \exists r.Y,\ \exists r.Y \sqsubseteq B,\ Y \sqsubseteq B,\ Y \sqsubseteq Y',\ Y' \sqsubseteq Y\}$. Then the set $\mathsf{Just}_{\mathcal{T}}(A \sqsubseteq B)$ consists of the two sets $\{A \sqsubseteq X,\ X \sqsubseteq \exists r.Y,\ \exists r.Y \sqsubseteq B\}$ and $\{A \sqsubseteq Y,\ Y \sqsubseteq B\}$.

Note that there can exist exponentially many justifications for a given TBox $\mathcal{T}$ and a concept inclusion $C \sqsubseteq D$. Hence, unlike with standard reasoning in $\mathcal{ELH}$, it is not possible to compute all justifications in polynomial time w.r.t. the size of the TBox in every case [2]. It is however still (theoretically) possible to compute *one* justification in polynomial time.

3 Tool Overview & Computation Techniques

Just v 0.1 is implemented in Java and it takes an $\mathcal{ELH}$-TBox $\mathcal{T}$ and two $\mathcal{EL}$ concepts C, D as input. The default operation mode of Just is then to compute and output the set $\mathsf{Just}_{\mathcal{T}}(C \sqsubseteq D)$. Alternatively, the tool can be instructed to search for one justification for $\mathcal{T} \models C \sqsubseteq D$ only.

We first focus on describing how the computation of justifications for inclusions between concept names w.r.t. normalised TBoxes is performed in Just. A TBox $\mathcal{T}$ is said to be in normal form if every axiom of $\mathcal{T}$ is of one of the following forms: $\prod_{i=1}^{n} A_i \sqsubseteq B$, $A \sqsubseteq \exists r.\bar{X}$, $\exists r.\bar{X} \sqsubseteq B$, or $r \sqsubseteq s$, where $n \geq 0$, A, B are concept names, and an expression of the form $\bar{X}$ stands either for the concept name X or for the $\top$-concept.

The computation of justifications for inclusions between concept names w.r.t. a normalised TBox $\mathcal{T}$ is performed by a saturation procedure that computes all the (minimal) derivations for inclusions of the form $A \sqsubseteq B$, $\top \sqsubseteq A$, or $\top \sqsubseteq \top$, for

[1] Justifications are also known as *MinAs* in the literature.

$$(\text{Ax})\ \frac{}{\bar{A} \sqsubseteq \bar{A}} \qquad (\text{AxTop})\ \frac{}{A \sqsubseteq \top} \qquad (\text{RoleAx})\ \frac{}{r \sqsubseteq r}$$

$$(\text{RoleSucc})\ \frac{r \sqsubseteq s}{r \sqsubseteq t} \quad \text{if } s \sqsubseteq t \in \mathcal{T}$$

$$(\text{Ex})\ \frac{\bar{X} \sqsubseteq \bar{Y} \qquad r \sqsubseteq s}{\bar{A} \sqsubseteq B} \quad \text{if } \bar{A} \sqsubseteq \exists r.\bar{X} \in \mathcal{T} \text{ and } \exists s.\bar{Y} \sqsubseteq B \in \mathcal{T}$$

$$(\text{Conj})\ \frac{\bar{A} \sqsubseteq X_1 \qquad \ldots \qquad \bar{A} \sqsubseteq X_n}{\bar{A} \sqsubseteq B} \quad \text{if } X_1 \sqcap \ldots \sqcap X_n \sqsubseteq B \in \mathcal{T} \text{ with } n \geq 0$$

$$(\text{Merge})\ \frac{\bar{A} \sqsubseteq \bar{X} \qquad \bar{X} \sqsubseteq \bar{B}}{\bar{A} \sqsubseteq \bar{B}}$$

Fig. 1. Calculus $\mathfrak{T}$

concept names A and B, using the calculus $\mathfrak{T}$ depicted in Fig. 1. The calculus $\mathfrak{T}$ is related to calculi used for consequence-based reasoning [6]. Note that expressions of the form $\bar{A}$ must be consistently instantiated (either with concept names or $\top$) in the premise and the conclusion of inference rules.

The axioms used in the application of the inference rules for generating a derivation for $X \sqsubseteq Y$ yield a subset of $\mathcal{T}$ from which the consequence $X \sqsubseteq Y$ logically follows. The minimal such axiom sets resulting from all the (minimal) derivations for $X \sqsubseteq Y$ are therefore the justifications for $X \sqsubseteq Y$ w.r.t. $\mathcal{T}$.

Example 2. Let $\mathcal{T}$ be defined as in Example 1. Then, for instance, the following two derivations Δ_1 and Δ_2 for the inclusion $A \sqsubseteq B$ can be generated w.r.t. $\mathcal{T}$ using the calculus $\mathfrak{T}$. We associate with each derivation Δ_i a set $\mathsf{Axioms}(\Delta_i)$ which consists of the axioms of $\mathcal{T}$ that were used in the inference rule applications occurring in Δ_i. First, Δ_1 is given as follows:

$$\dfrac{\dfrac{(\text{Ax})\ \dfrac{}{A \sqsubseteq A}}{(\text{Conj})\ \dfrac{A \sqsubseteq A}{A \sqsubseteq Y}} \qquad \dfrac{\dfrac{}{Y \sqsubseteq Y}\ (\text{Ax})}{\dfrac{Y \sqsubseteq Y}{Y \sqsubseteq B}\ (\text{Conj})}}{A \sqsubseteq B}\ (\text{Merge})$$

We have $\mathsf{Axioms}(\Delta_1) = \{A \sqsubseteq Y, Y \sqsubseteq B\}$. The derivation Δ_2 can be depicted as

$$\dfrac{\dfrac{(\text{Ax})\ \dfrac{}{A \sqsubseteq A}}{(\text{Conj})\ \dfrac{A \sqsubseteq A}{A \sqsubseteq Y}} \qquad \dfrac{\dfrac{\dfrac{(\text{Ax})\ \dfrac{}{Y \sqsubseteq Y}}{(\text{Conj})\ \dfrac{Y \sqsubseteq Y}{Y \sqsubseteq Y'}} \qquad \dfrac{\dfrac{}{Y' \sqsubseteq Y'}\ (\text{Ax})}{\dfrac{Y' \sqsubseteq Y}{Y \sqsubseteq Y}\ (\text{Conj})}}{Y \sqsubseteq Y}\ (\text{Merge})}{Y \sqsubseteq B}\ (\text{Conj})}{A \sqsubseteq B}\ (\text{Merge})$$

with $\mathsf{Axioms}(\Delta_2) = \{A \sqsubseteq Y, Y \sqsubseteq B, Y \sqsubseteq Y', Y' \sqsubseteq Y\} \supsetneq \mathsf{Axioms}(\Delta_1)$.

Following the example of derivation Δ_2, it is possible to construct infinitely many derivations for the inclusion $A \sqsubseteq B$ from the TBox $\mathcal{T}$ (as defined in Example 1). However, one can prove that for finding all justifications it is sufficient to construct so-called *admissible* derivations Δ only in which every subderivation Δ' of an inclusion α does not contain an occurrence of α as a premise in Δ'. It is easy to see that there can only exist finitely many admissible derivations w.r.t. a normalised TBox $\mathcal{T}$ for a given inclusion α.

To compute justifications w.r.t. general TBoxes, JUST keeps track of which original axiom corresponds to which normalised axiom. The justifications for a consequence $C \sqsubseteq D$ w.r.t. a general TBox $\mathcal{T}$ are then generated from the justifications for $C \sqsubseteq D$ w.r.t. the normalisation of $\mathcal{T}$ by successively replacing every normalised axiom with all the axioms from which it originates. The minimal sets obtained in that way are the justifications w.r.t. the general TBox.

Example 3. Let $\mathcal{T} = \{A \sqsubseteq B \sqcap \exists r.\top,\ A \sqsubseteq B \sqcap \exists s.\top\}$ and let $\mathcal{T}_N = \{A \sqsubseteq B,\ A \sqsubseteq \exists r.\top,\ A \sqsubseteq \exists s.\top\}$ be the normalisation of $\mathcal{T}$. Then $\mathsf{Just}_{\mathcal{T}_N}(A \sqsubseteq B) = \{\{A \sqsubseteq B\}\}$. As the axiom $A \sqsubseteq B$ in $\mathcal{T}_N$ originates from both axioms in $\mathcal{T}$, we obtain that $\mathsf{Just}_{\mathcal{T}}(A \sqsubseteq B)$ consists of the two sets $\{A \sqsubseteq B \sqcap \exists r.\top\}$ and $\{A \sqsubseteq B \sqcap \exists s.\top\}$.

The computation of justifications for inclusions of the form $C \sqsubseteq D$ w.r.t. $\mathcal{T}$, where C and D are not necessarily concept names, can be done analogously by computing justifications for $A_C \sqsubseteq A_D$ w.r.t. $\mathcal{T} \cup \mathcal{X}$, where $\mathcal{X} = \{A_C \sqsubseteq C, D \sqsubseteq A_D\}$ for fresh concept names A_C and A_D. The justifications for $C \sqsubseteq D$ are then the minimal sets obtained from the justifications for $A_C \sqsubseteq A_D$ after removing the axioms contained in $\mathcal{X}$.

Example 4. Let $\mathcal{T} = \{A \sqsubseteq B\}$ and let $\mathcal{X} = \{A_C \sqsubseteq A \sqcap Y, B \sqsubseteq A_D\}$. Then $\mathsf{Just}_{\mathcal{T} \cup \mathcal{X}}(A_C \sqsubseteq A_D) = \{\{A_C \sqsubseteq A \sqcap Y, B \sqsubseteq A_D, A \sqsubseteq B\}\}$ and $\mathsf{Just}_{\mathcal{T}}(A \sqcap Y \sqsubseteq B) = \{\{A \sqsubseteq B\}\}$.

Similarly to [9], JUST extracts a reachability-based module for $X \sqsubseteq Y$ first before starting to compute all the (minimal) derivations for $X \sqsubseteq Y$. Moreover, we used techniques from resolution-based theorem provers to obtain an acceptable performance of our tool in practice. In particular, whenever a derivation Γ' for an inclusion α was generated, another derivation Γ for α had been obtained previously, and all the axioms associated with Γ were contained in the set of axioms associated with Γ', then Γ' was discarded. Such a deletion strategy corresponds to forward subsumption deletion in resolution theorem provers. We also employed a technique equivalent to backward subsumption deletion.

The computation of a single justification is currently implemented by stopping the saturation process after a derivation for the target inclusion has been found.[2] As the axioms occurring in this derivation might not represent a justification yet, superfluous axioms are identified by checking for each axiom whether the target inclusion still follows (using the reasoner ELK [7]) after the considered axiom has been removed.

[2] Note that in JUST v 0.1 the saturation step is not guaranteed to finish in polynomial time w.r.t. the size of the input TBox, even when only *one* justification is computed.

Table 1. Metrics of the Ontologies Used in the Experiments

Ontology	# Axioms	# Concept Names	# Role Names
NCI	117 583	105 105	92
SNOMED CT	354 601	291 144	57
GALEN-OWL	46 457	23 136	949

4 Experimental Evaluation

For our experiments we picked three ontologies that are typically considered to pose different challenges to DL reasoners and that are expressed mainly in $\mathcal{ELH}$: version 13.12e of the NCI thesaurus,[3] the January 2010 international release of SNOMED CT, and the GALEN-OWL ontology.[4] In the case of NCI all the 152 axioms that fell outside the considered $\mathcal{ELH}$ fragment were first removed from the ontology. The number of axioms, concept names, and role names in the resulting ontologies is shown in Tbl. 1. For each of the three ontologies $\mathcal{T}$ we then randomly selected 1000 inclusions between concept names, $A \sqsubseteq B$, such that $\mathcal{T} \models A \sqsubseteq B$ holds. In a first set of experiments we used JUST and the algorithm for computing all justifications implemented in the OWL-API [3,4] (using the reasoner FaCT++[5] [10]) to compute all the justifications for each selected inclusion w.r.t. the respective ontology. In a second series of experiments we instructed JUST to only search for one justification for each considered inclusion. All experiments were conducted on a PC equipped with an Intel i5-2500K CPU running at 3.30GHz and with 16 GiB of main memory. An execution timeout of 10 CPU minutes was imposed on each problem.

The results obtained for computing all justifications are shown in Tbl. 2. The first column indicates which ontology was used. The next five columns then show the results obtained with JUST, whereas the last two columns refer to the OWL-API implementation. More precisely, the second and seventh column indicate the number of successful computations within the given time limit by the respective implementations. The average and the maximal number, as well as the maximal size of the justifications as computed by JUST are shown in the next three columns. The sixth and eighth columns contain the average CPU time

[3] `http://evs.nci.nih.gov/ftp1/NCI_Thesaurus`

[4] `http://owl.cs.manchester.ac.uk/research/co-ode/`

[5] Note that at the time of writing it was not possible to use the reasoner ELK in combination with the OWL-API implementation for computing justifications.

Table 2. Results Obtained for Computing All Justifications

	JUST					OWL-API	
Ontology	# Success.	Avg. #	Max. #	Max. Size	Time (s)	# Success.	Time (s)
NCI	954/1000	1.165	32	10	34.895	577/1000	231.532
SNOMED CT	876/1000	2.622	127	28	57.405	16/1000	395.882
GALEN-OWL	511/1000	1.315	4	16	8.740	0/1000	-

Table 3. Results Obtained for Computing One Justification using JUST

Ontology	# Success.	Max. Size	Time (s)
NCI	972/1000	10	36.827
SNOMED CT	951/1000	28	53.473
GALEN-OWL	978/1000	22	71.759

required by the respective implementations for the successful computations over each considered set of inclusions.

Regarding the computation of all justifications with JUST, the lowest number of successful computations were observed for GALEN-OWL, despite it being the smallest ontology of the corpus. The largest size, average & maximal number of justifications, as well as the longest average computation times involving JUST, were found for SNOMED. No computation succeeded within the allocated time for experiments involving GALEN-OWL using the OWL-API implementation. In general, we observed fewer timeouts and shorter average computation times with JUST than with the OWL-API.

The results obtained for computing only one justification using JUST are given in Tbl. 3. Analogously to Tbl. 2, the number of successful computations and the maximal size of the computed justifications are given in the second and third columns. The last column shows the average CPU time required to compute one justification.

One can see that fewer timeouts occurred when only one instead of all justifications were computed with JUST. The least number of successful computations was now observed for SNOMED, whereas only 22 computations involving GALEN-OWL did not succeed within the given time limit. The longest average computation times were reported for GALEN-OWL, and the largest justification was again computed for SNOMED CT.

In all the successful computations JUST required at most 14.13 GiB of main memory.

5 Conclusion

We presented the tool JUST for computing either one, or all the justifications for general concept inclusions w.r.t. ontologies formulated in $\mathcal{ELH}$. Our experimental evaluation showed that JUST is capable of finding all the justifications for consequences in many practical cases within a reasonable time, even when a large number of justifications exist.

As future work we aim to implement an extended calculus which would allow it to compute justifications for more expressive description logics. Our tool could also benefit from a more goal-oriented construction of derivations and from an improved module extraction procedure that yields smaller modules. It would also be interesting to implement an incremental computation of justifications, which would permit it to generate as many justifications as requested by the user. Finally, we also plan to perform a more extensive comparison of the performance of our tool against other approaches for computing justifications.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2nd edn. (2007)
2. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic $\mathcal{EL}^+$. In: Proceedings of the 30th Annual German Conference on AI (KI 2007). Lecture Notes in Computer Science, vol. 4667, pp. 52–67. Springer (2007)
3. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semantic Web 2(1), 11–21 (2011)
4. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Proceedings of the 6th International Semantic Web Conference & 2nd Asian Semantic Web Conference (ISWC 2007 & ASWC 2007). Lecture Notes in Computer Science, vol. 4825, pp. 267–280. Springer (2007)
5. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.A.: Debugging unsatisfiable classes in OWL ontologies. Journal of Web Semantics 3(4), 268–293 (2005)
6. Kazakov, Y.: Consequence-driven reasoning for Horn SHIQ ontologies. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09). pp. 2040–2045 (2009)
7. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK: From polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. Journal of Automated Reasoning 53(1), 1–61 (2014)
8. Suntisrivaraporn, B.: Finding all justifications in SNOMED CT. ScienceAsia 39(1), 79–90 (2013)
9. Suntisrivaraporn, B., Qi, G., Ji, Q., Haase, P.: A modularization-based approach to finding all justifications for OWL DL entailments. In: Proceedings of the 3rd Asian Semantic Web Conference (ASWC 2008). Lecture Notes in Computer Science, vol. 5367, pp. 1–15. Springer (2008)
10. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR 2006). Lecture Notes in Computer Science, vol. 4130, pp. 292–297. Springer (2006)
11. Zhou, Z., Qi, G., Suntisrivaraporn, B.: A new method of finding all justifications in OWL 2 EL. In: Proceedings of the 2013 IEEE/WIC/ACM International Conferences on Web Intelligence (WI 2013). pp. 213–220. IEEE Computer Society (2013)