Inference Inspector: Improving the Verification of Ontology Authoring Actions

Nicolas Matentzoglu, Markel Vigo,
Caroline Jay, Robert Stevens

{firstname.secondname}@manchester.ac.uk

Abstract

Ontologies are complex systems of axioms in which unanticipated consequences of changes are both frequent, and difficult for ontology authors to apprehend. The effects of modelling actions range from unintended inferences to outright defects such as incoherency or even inconsistency. One of the central ontology authoring activities is verifying that a particular modelling step has had the intended consequences, often with the help of reasoners. For users of Protégé, this involves, for example, exploring the inferred class hierarchy.

This paper provides evidence that making entailment set changes explicit to authors significantly improves the understanding of authoring actions regarding both correctness and speed. This is tested by means of the Inference Inspector, a Protégé plugin we created that provides authors with specific details about the effects of an authoring action. We empirically validate the effectiveness of the Inference Inspector in two studies. In a first, exploratory study we determine the feasibility of the Inference Inspector for supporting verification and isolating authoring actions. In a second, controlled study we formally evaluate the Inference Inspector and determine that making changes to key entailment sets explicit significantly improves author verification compared to the standard static hierarchy/frame-based approach. We discuss the advantages of the Inference Inspector for different types of verification questions and find that our approach is best suited for verifying added restrictions where no new signature, such as class names, is introduced, with a 42% improvement in verification correctness.

Keywords: OWL, ontologies, human computer interaction, ontology engineering, ontology authoring, reasoning

1. Introduction

Ontologies are explicit conceptualisations of a domain and are widely applied in biology, health-care and the public domain [5, 27, 35]. Ontologies are typically represented in a formal representation language such as the Web Ontology Language (OWL), the Open Biomedical Ontologies format (OBO) or the RDF Schema language (RDFS).[1] The central advantage of using such formalisms is their well-defined semantics. Generic reasoning systems can be used to access knowledge in the ontology that is only implied, i.e. not explicitly stated, allowing richer answers to queries, the identification of inconsistent knowledge and improved management of large terminologies through definition-oriented development. There is strong, if primarily anecdotal, evidence that building ontologies using OWL is difficult and error-prone.

Attempts have been made to quantify this difficulty [12], but there remain many unanswered questions about the cognitive challenge of various ontology authoring tasks such as exploration or modelling. The complexity of OWL [43] can lead to axioms that do not reflect the intentions of the author. Furthermore, the lack of understanding for many of the OWL 2 features (in particular, but not only, by domain experts) can result in unintended inferences, which are often not made explicit by the authoring tool, and even if they are, are rarely communicated to the author clearly. An interview study recently revealed that many ontology experts *frequently* run the reasoner, sometimes *after every modification*, to detect errors such as unsatisfiable classes and to prevent the spread of errors [37]. Participants of that study also felt that the change evaluation phase, i.e. the phase that determines whether a modelling action had the intended consequences, is not well supported by state of the art development tools. Some ontology authors use DL queries, generated on the fly, to do 'spot

[1] OBO and RDFS can be seen as syntactic variations of OWL [8].

checks', others work with competency questions that are crafted upfront to automatically verify the correctness of a change. As the conceptual model of an ontology is, however, not always known upfront, competency question-based approaches, perhaps best compared with unit tests in software engineering, are of particular utility later in the engineering process, where their coverage of the ontology depends on the user's diligence. Consequently, we need the user interface to reduce the perceived complexity of ontologies, support their evaluation, and help prevent or detect errors.

In this work, we are concerned with improving the evaluation of modelling actions. We call the task of evaluating whether a particular modelling action has had the desired effect "verification". Verification is a key sub-process of ontology authoring that involves conducting a set of tests, for example to make sure that a definition of a class works as intended and that no unsatisfiable classes were introduced [4]. We distinguish in our analysis between different types of verification problems (or questions), which we selected based on our extensive experience with teaching OWL and ontology authoring [34]. Examples of verification problem types are verification of tightened restrictions (for example when adding an existential restriction[2] to a class), verification of fixing an unsatisfiability (did the latest change remove the unsatisfiability?) or verification of a definition (are individuals correctly inferred to be members of that class? Does the class have the expected sub-classes?). An enumeration and definition of such types is necessarily subjective and incomplete; we motivate our selection in the context of the methodological discussion. When developing ontologies with the widely used Protégé ontology engineering environment, the verification step is typically realised by invoking the reasoner and exploring the implicit knowledge in the ontology [37], for example by making sure that a particular class has the expected position in the inferred class hierarchy or a freshly introduced property domain restriction results in the expected individual type inferences. We call this approach *static hierarchy/frame-based* (SHFB), where "static" refers to the fact that the inferred hierarchy only reflects a state, without any indication as to how this state relates to the latest modelling action.

We propose a method to improve verification by presenting changes to finite entailment sets, i.e. sets of axioms of a particular form that are implied by the ontology [1]. A prominent example of a finite entailment set is the set of all implied subclass (or disjointness) relations between classes in the ontology. We explore the hypothesis that making changes to key entailment sets explicit improves verification compared to the static hierarchy/frame-based approach.

Our contributions are as follows:

- We developed the Inference Inspector, a novel Protégé plugin that alerts the author to the changes to key entailment sets that have occurred as a consequence of a modelling action.
- We conducted an exploratory study to evaluate our Inference Inspector prototype. We find that our approach is better suited to tasks that involve changing definitions or adding restrictions on existing entities, and less well suited for tasks that involve the introduction of new entities, compared to the SHFB approach.
- We conducted a laboratory experiment that confirms our hypothesis. We find that making entailment set changes explicit improves the understanding of consequences both in terms of correctness and speed, and is rated as the preferred way to inspect the consequences of changes, compared to the SHFB approach.

2. Background and Related Work

Ontology authoring is the creation and maintenance of ontology artefacts constructed using a formal knowledge representation language such as OWL, OBO or RDFS. We view an ontology $\mathcal{O}$ as a set of axioms, with $\alpha \in \mathcal{O}$ being an axiom in $\mathcal{O}$. The signature of $\mathcal{O}$, denoted $\widetilde{\mathcal{O}}$, is the set of logically relevant entities, i.e. classes, object properties, data properties and individuals,[3] across all axioms in $\mathcal{O}$. Given a language $\mathcal{L}$ and an OWL 2 ontology $\mathcal{O}$, the $\mathcal{L}$-entailment set of $\mathcal{O}$, written $\mathcal{E}(\mathcal{O}, \mathcal{L})$, is the set of all axioms in $\mathcal{L}$ that are entailed by $\mathcal{O}$ (entailment set). One of the most important entailment sets for ontology authoring is the set of atomic subsumptions, i.e. the language $\mathcal{L}$ that allows us to build axioms of the form $SubClassOf(A, B)$ for all valid combinations of A and B in signature $\widetilde{\mathcal{O}} \cup \top \cup \bot$. More precise definitions of the entailment sets relevant to this work can be found in Appendix A.

Typical ontology authoring activities include, but are not limited to, the creation of axioms or annotations.

[2] I.e. adding an axiom of the form $SubClassOf(A, R \text{ some } B)$ for a class A.

[3] For our purposes, this excludes in particular annotation properties and datatypes, which are considered entities in OWL 2, but are not of interest to logical reasoning.

For a detailed discussion of ontology authoring activities see [40]. While ontology authoring is increasingly performed in a programmatic fashion, a large number of ontologies have been built using ontology authoring environments such as Protégé [19] and WebProtégé [36]. Moreover, based on our experience, even if ontologies are created in a programmatic fashion, they are often checked for defects in a visual authoring environment.

Research on ontology authoring has experienced a resurgence in recent years [37, 40, 41], due at least in part to the increased availability of change-logs for ontology development. WebProtégé, for example, produces detailed change-logs, which can form the basis of a rich and informative analysis of ontology authoring activities [41]. The work presented here builds on a series of investigations into the ontology authoring process [37, 38, 40]. The aim of this body of work is to improve our understanding of ontology authoring processes, and in particular to identify typical authoring styles and workflows that will help tool developers to improve their support of the authoring process. We have identified a number of difficulties shared across ontology developers that occur during ontology development, and have found that Protégé does not support the needs of current authors [37, 38]. In particular, ever more sophisticated ontology modelling patterns make the verification of modelling actions difficult. For example, the combination of a $SubClassOf$ axiom such as $SubClassOf(A, R \text{ some } B)$ with a $ObjectPropertyDomain(R, C)$ can influence the class hierarchy, at least for a significant proportion of the OWL 2 users, unexpectedly.[4] The fact that unintended consequences such as the introduction of unsatisfiable classes, broken definitions (that result in inaccurate classifications) or inaccurate inferences on the data level (ABox) are often difficult to spot was one of the core incentives for this work. A specially modified version of Protégé that collects interaction events silently during ontology authoring [40], Protégé4US, enabled us to study ontology authoring workflows and derive a number of well-founded design suggestions for authoring tools [40]. One of these was *making the changes to the inferred hierarchy explicit*— another major incentive for developing the Inference Inspector.

Developing better support for ontology authoring is a long-standing challenge spanning a number of domains including visualisation [16] and debugging [31]. Ontology visualisations, concerned with developing (scalable) visualisations of ontology concepts, their instances and inter-relations, support ontology authors in exploration tasks (such as verification or familiarisation) and making portions of the ontology (such as justifications) easier to understand, for example using graphs or graph-like structures, natural language [14], diagrammatic representations, such as crop-circles [42] or, for simpler ontologies, graphs [13]. While the majority of ontology visualisation focuses on graph-like representations (2D and 3D) and tree-like hierarchical structures [16], efforts are being made towards presenting ontologies in the form of natural language [22] or axioms [43]. It is clear that no particular form of representation will cater to all needs, or indeed to all author preferences, and that the field should continue investigating which representation is most suitable for which authoring tasks, or types of author. Our work explores the use of an axiomatic representation for improving authoring because of its suitability for representing changesets, but we do not claim that it is the best way, rather that for a well-defined set of problems, it is better than the dominant existing approach.

Ontology debugging is concerned with developing tools for identifying and correcting defects of an ontology. Defects range from broken syntax and language violations, such as OWL profile violations [23], to broken definitions, unsatisfiable classes and ontology inconsistency [10, 15]. One principal technique for addressing the latter (logical) violations are justifications [11], minimal subsets of an ontology from which a (possibly unintended) inference follows. Given an entailment α and an ontology O, a justification $\mathcal{J}_O^\alpha$ is a minimal subset of O that entails α. Research on justifications includes determining their cognitive complexity [12], making them more accessible to ontology authors [28] or using them for reasoner verification [21]. Other approaches to ontology debugging include evaluation tools such as the Ontology Pitfall Scanner [33], the detection of emerging flaws throughout the entire version history of an ontology through axiom dynamics analysis [3] and a unified approach to ontology debugging and ontology alignment [13].

The existing tool support for ontology authoring activities is still largely inadequate [37]. An example of an early study that established the necessity of presenting explanations for entailments and reporting errors adequately in the context of knowledge representation (KR) systems was McGuiness et al. [25]. Ontology authoring tools have continued to receive poor usability ratings [6, 20, 37] over the last 20 years. Examples of unmet demands from users include the ability to compare different versions of the ontology [6] and inadequate debugging support [37]. In particular, making the consequences of modelling actions explicit beyond simply

[4] Namely, by causing A to be a subclass of C.

identifying that a defect exists has received little attention. The most significant effort to achieve this came from Denaux et al. [4],[5] who developed a system that provides interactive semantic feedback directly after a change to the ontology. They suggest 6 categories of semantic feedback from the ontology engineering environment, given a single axiom α being added to the ontology: (A) α was already asserted ($\alpha \in \mathcal{O}$), (R) α was not asserted, but could be inferred ($\alpha \notin \mathcal{O}; \mathcal{O} \models \alpha$), (I) α causes the ontology to be inconsistent ($\alpha \cup \mathcal{O} \models \top \sqsubseteq \bot$), (N+) α is novel, and the addition results in new implications ($\mathcal{O} \not\models \alpha; \alpha \cup \mathcal{O} \not\models \top \sqsubseteq \bot; \alpha \cup \mathcal{O} \models \eta$, with $\eta \neq \alpha$ and $\mathcal{O} \not\models \eta$, (N) α is novel, and the addition does not result in new implications ($\mathcal{O} \not\models \alpha; \alpha \cup \mathcal{O} \not\models \top \sqsubseteq \bot; \forall \eta$ with $\alpha \cup \mathcal{O} \models \eta$, $\mathcal{O} \models \eta$ and (U) α causes a concept in the ontology to become unsatisfiable ($\mathcal{O} \not\models \alpha; \alpha \cup \mathcal{O} \not\models \top \sqsubseteq \bot; \exists A \in \widetilde{\mathcal{O}}$ with $\mathcal{O} \not\models A \sqsubseteq \bot$ and $\alpha \cup \mathcal{O} \models A \sqsubseteq \bot$).[6] While the work by Denaux et al. inspired us to produce a better feedback mechanism, it differs in three fundamental aspects to the research presented here:

- Only additions are modelled, i.e., where an engineer adds an axiom to the ontology; we cover both additions and removals.
- Changes comprise a single axiom; we model both single axiom changes, and sets of additions and removals.
- The authors evaluate their approach using a task-based setting similar to our exploratory study, and find that the feedback was generally considered helpful, but no formal evaluation was conducted to find out whether the feedback actually led to more accurate modelling; we conduct a controlled study to formally evaluate the efficacy of our approach in terms of modelling speed and accuracy.

Only providing feedback when the reasoner is run returns the responsibility for asking for feedback to the engineer and keeps the interface responsive (reasoning is not required after every step), but also comes with a caveat: given a set of changes, it may no longer be possible to attribute a particular inference (either lost or gained) to a particular change, thereby returning the burden of identifying the erroneous change to the engineer. We believe, however, that the gain in responsiveness is worth this caveat, and we can cover some of these shortcomings using justifications, as explained in the next section.

[5]The tool is available online at `https://sourceforge.net/projects/entendre/`

[6]We have changed the notation slightly to suit our own taste, but the meaning remains the same.

3. Inference Inspector: Making the Consequences of Modelling Actions Explicit

We present the Inference Inspector, a Protégé plugin for making the consequences of modelling actions in an ontology explicit. The Inference Inspector is implemented as a plugin for Protégé 5 (5.0.0 at the time of writing). We consider ontologies to be represented in OWL 2 DL, unless otherwise stated. A *modelling action* is defined as a non-empty set of changes $\mathcal{CH}$. A *change* can either be a removal of an axiom α, denoted R_α, or an addition, denoted A_α. For example, given the addition of an axiom α_1: SubClassOf(A, B) and the removal of another axiom α_2: SubClassOf(A, C), the modelling action is defined as $\mathcal{CH} : \{A_{\alpha_1}, R_{\alpha_2}\}$. Axiom *modifications* are always treated as an addition of the revised axiom and a removal. In the previous example, the user might have decided that A should not be subsumed by B, but by C instead, changing the existing SubClassOf(A, B) to SubClassOf(A, C). The Inference Inspector makes changes to predefined key entailment sets explicit. A change to an entailment set is defined as follows. Given an ontology $\mathcal{O}$, a previous version of the ontology $\mathcal{O}'$ and a finite entailment set $\mathcal{E}$, the difference between the respective entailment set of $\mathcal{O}$ and $\mathcal{O}'$, $\mathcal{E}_\mathcal{O} \setminus \mathcal{E}_{\mathcal{O}'}$ is called the set of *added inferences w.r.t.* $\mathcal{E}$, and the difference between the entailment set of $\mathcal{O}'$ and $\mathcal{O}$, $\mathcal{E}_{\mathcal{O}'} \setminus \mathcal{E}_\mathcal{O}$ is called the set of *removed inferences w.r.t.* $\mathcal{E}$. In the following, we often refer to axioms that add a constraint on an OWL entity as a "restriction". For example, $SubClassOf(A, B \text{ and } C)$ restricts A (namely by making all instances of A instances of B and C), and $OWLObjectPropertyDomain(R, A)$ restricts R (namely by making all instances related to something via R instances of A). Consequently, we say "loosening" a restriction when we either (1) make it less strong, for example by changing an axiom $SubClassOf(A, B \text{ and } C)$ to $SubClassOf(A, B)$ or (2) by removing it altogether, and "tightening" for the respective opposite.

When using the Inference Inspector, a snapshot of the current ontology is created after each reasoner run, including its inferences (i.e. the entailment sets selected). By default, users are presented with the consequences of their most recent modelling action (Figure 1). We define the scope of a single modelling action as the set of all changes that were applied to the ontology between the latest and the previous run of the reasoner. For example, after running the reasoner, the ontology author might add three axioms and remove one. When the user runs the reasoner again, they will be presented with the entailments added and lost since the previous run of the reasoner. From an implementation perspec-

tive, this is realised by computing the set difference in accordance with the definitions given in the beginning of this section. The Inference Inspector allows the user to compare the current state of the ontology to any snapshot created previously. The first snapshot is always the empty ontology: this means that comparing the current version of the ontology with the empty ontology will always show all inferences (short of those that are explicitly hidden by the user). By default, inferences of critical importance (P1 in Figure 1) are always shown (see Section 3.2), no matter which snapshot forms the basis for comparison, but this feature can be switched off if only the latest changes are of interest. For all entailments, justifications can be computed on demand. In the following section, we will discuss how we selected relevant entailment sets and how we present them to the user.

3.1. Entailment Set Selection

There are a number of approaches for selecting appropriate entailment sets for presentation, including syntactic, semantic and pragmatic [32]. At present, however, there is no conclusive evidence (in fact, no evidence at all that we know of), as to which entailment sets are most useful for ontology authoring. Our requirements are practical: the entailments shown should help the user to verify their modelling actions and not be too costly to compute.[7] To achieve this, we believe that the entailment set should be indicative of erroneous or correct modelling, of general interest, and easily understandable by a typical user.

In order to be *indicative of erroneous modelling*, the presented entailments must be verifiable against modelling intentions. Since we cannot directly access the user's modelling intentions, we make a number of simplifying assumptions. Firstly, we consider unsatisfiable classes or ontology inconsistency as bugs, which any user aims to avoid. Secondly, we assume that the majority of users have a mental model of the hierarchical structures of their ontology, including a model of class disjointness, and intend to keep the ontology consistent with that mental model. In other words, we assume that the ontology author knows, for a concept they are modelling, where in the hierarchy it should be situated and which individuals should be members of it, as well as whether it is disjoint from another concept in the domain. Therefore, we primarily serve the modelling intentions of avoiding bugs while producing hierarchies consistent with that mental model. We acknowledge that this assumption is not universally true, as it is, for example, unlikely that any one author of the gene ontology [2] knows all subsumptions between all the concepts it covers. Furthermore, there are other relevant axes that are not covered by our approach, such as partonomy or any class level patterns that are based on object properties, such as existential restrictions. We do, however, believe that the subsumption relation is of central importance in the majority of cases, which is also confirmed by our finding that users look at the class hierarchy 45% of the time they spend editing ontologies with Protégé [40].

The presented entailments must also be *easily understandable* by a user, i.e., we do not want to replace a cognitively demanding search, such as a lookup in a large hierarchy, by a cognitively demanding parse, such as an axiom involving deeply nested class expressions. Therefore, our approach only considers entailments that involve named entities, and avoid those that involve complex class expressions such as existential restrictions.

Axioms of *general interest* are those that are used for modelling in practice. Across BioPortal [29], we found that 79.56% of all axioms measured correspond to subsumptions (classes and properties), 6% to class assertions, 5.61% to object property assertions, 1% to disjoint classes,[8] 0.74% to equivalences (classes and properties), 0.07% to property characteristics and the rest (7.02%) to other axiom types. More than half of the ontologies (166 out of 329) contain no other axiom types corresponding to the entailment sets we selected. This kind of breakdown only gives us an indication of what might be of general interest to our hypothetical user; if users model predominantly on the class hierarchy level, this might be an indication that they are also interested in inferences on that level. Of course, this is only a fairly weak indication, as some axiom types might occur rarely, but have a large logical impact on the ontology (for example a functional property that causes all individuals to become disjoint). Furthermore, the above breakdown ignores the complexity of the used class expressions: only 44% of all subsumption axioms correspond to atomic subsumption, and only 1% of the equivalence axioms are atomic. Materials and methods for this survey can be found in Appendix D.

Lastly, determining the entailment sets should not be too *computationally expensive*. While the theoretical

[7] *Costly* in the empirical sense of the word - time consuming given a realistic ontology and a general OWL 2 reasoner.

[8] Lower bound: covers DisjointClasses(A, B), ignores $SubClassOf(A, not(B))$.

worst case complexity for OWL classification is 2NEXPTIME [17], reasoning is empirically robust [9] (i.e. performs classification in "reasonable" time) and performs classification and consistency well for a wide variety of inputs [30]. Computing the full set of disjoint classes, on the other hand, can be more computationally intensive in practice because reasoners do not implement efficient algorithms for this task. We do, however, consider the set of disjoint classes as highly indicative of modelling error and (at least compared to other axiom types such as the various property characteristics) easy to understand. We, therefore, allow the user to determine whether computing the disjointness relation is worth their while.[9]

We consider the following (groups of) entailment sets: (1) atomic subsumptions between classes, and object and data properties; (2) equivalences between classes, and object and data properties; (3) object property characteristics; (4) disjointness between class names; (5) class assertions and; (6) object property assertions. While (1), (2), (4) and (5) directly correspond to the considerations above, (3) and (6) do not. We include object property assertions in our solution in order to provide a mechanism that allows the user to check whether sub-object property chains (and the object property hierarchy) work as intended. The reason for including object property characteristics was that our extensive experience teaching novice and advanced OWL users has shown that the inheritance or non-inheritance of object property characteristics up or down the object property hierarchy is extremely difficult to understand. For example, making an object property functional makes all its children's properties functional, while the same is not true for transitivity.

Although the entailment sets considered here are finite,[10] they are potentially large. For example, the set of atomic class subsumptions is in the worst case quadratic to the size of the classes in $\widetilde{O}$ (all classes are equivalent). To further reduce the amount of information shown to the user, for the three atomic subsumption entailment sets, we consider the transitive reduct, i.e. we query the reasoner only for direct subsumptions. A slightly extended discussion, size and definitions of the entailment sets used by the Inference Inspector can be found in Appendix A.

[9] Similar to Protégé, which allows restricting reasoning to particular entailment sets to increase performance.

[10] Since we do not consider complex class expressions, the size of the entailment set is bound by the size of the ontology signature.

3.2. Presenting Entailment Set Changes

In order to reduce the number of entailment set changes presented to the user, we have essentially three options: (1) ordering inference by relevancy, (2) grouping similar inferences together and (3) filtering out less interesting inferences [32].

To order the presented inferences by relevancy, the Inference Inspector implements a configurable system for inference prioritisation. We allow the user to assign a priority to an item from a list of pre-defined inference patterns, such as direct and indirect subsumptions or direct inferred assertions. Currently, we have implemented five priority levels and 11 inference patterns. Both priority levels and inference patterns were chosen based on the authors' extensive experience with ontology authoring. To increase the utility of the Inference Inspector, both will be developed further in the future. For the validity of our experimental results, this limitation is harmless, as a better, more comprehensive set of patterns or priority levels could only improve verification performance. The priority levels range from critically important (ontology defects such as unsatisfiability) to unimportant (e.g. asserted axioms). For a more detailed breakdown of the priority levels see Appendix C. By default, the Inference Inspector orders the presented consequences by priority, but it is also possible to order them lexically or by axiom type.

Ordering the potentially large number of entailments presented to the user is sometimes not enough. In particular, inferences on the ABox (individual) level can be extremely numerous. We, therefore, employ a grouping strategy for object property assertion axioms and class assertions [32]. By default, we group all axioms of the type ClassAssertion(a,X), where X is a particular class name in the ontology, and all axioms of the type ObjectPropertyAssertion(a, b, R), where R is a particular object property name in the ontology, by showing only one (random) exemplar for each X and R respectively. For very large ontologies the list of inferences can be further narrowed down by restricting it to particular entities in the ontology.

Lastly, the user may filter the inferences presented, by either: (1) showing only inferences related to the currently selected entity in Protégé or (2) showing only inferences involving entities manually selected in a special entity selector panel. An important caveat of the Inference Inspector implementation is that it relies on the correctness of the reasoner. Reasoners tend to be correct, but as they are complex software artefacts, bugs and incomplete implementations of the specification cannot be avoided [21]. Furthermore, a given reasoner

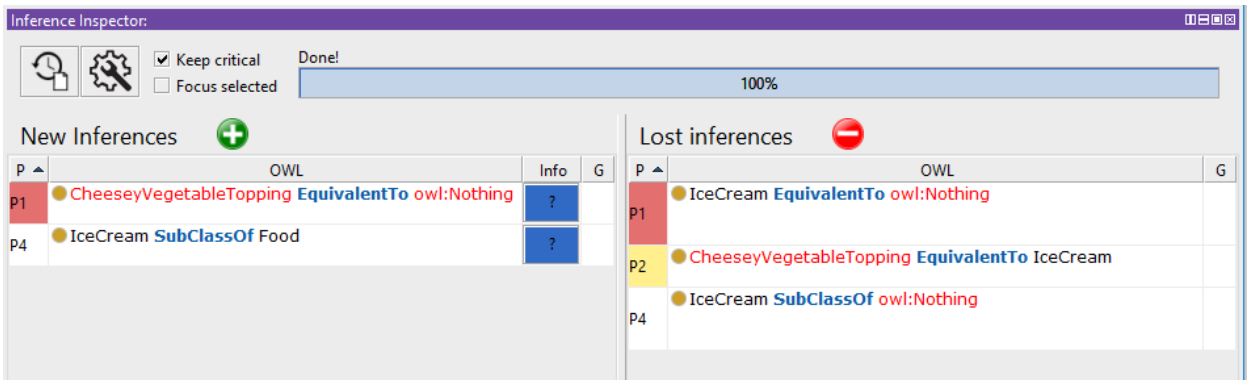


Figure 1: A snapshot of the Inference Inspector after the removal of an axiom that made IceCream unsatisfiable. Left: we can see the new position of IceCream in the class hierarchy. Right: we can see respective lost inferences. P1 (Critical), P2 (important) and P4 (not important) are priority levels (P).

may not support the inference of all the entailment sets considered by the Inference Inspector. For a discussion of additional features of the Inference Inspector see Appendix B.

4. Exploratory Study and Prototype Evaluation

We conducted an initial, exploratory study to evaluate our approach and the Inference Inspector tool. This first study was performed in the context of an advanced OWL modelling tutorial and was intended to evaluate our prototype and determine the modelling actions where authors may benefit from using the Inference Inspector.

4.1. Materials and Methods

Goals. The main goals of this study were to evaluate the Inference Inspector prototype and determine those modelling actions where our approach is likely to outperform existing solutions. The evaluation was designed to be broad and involved rating the Inference Inspector for perceived usefulness and responsiveness, as well as providing feedback on the user interface. From the results, we identified modelling actions for which our approach showed evidence of being beneficial, and used this information to design tasks for the second experiment (Section 5.1).

Participants. 15 intermediate users of Protégé were recruited in the context of a two-day advanced OWL tutorial (see `http://ow.ly/pK8P300x9wq`). Most participants had successfully completed a beginner level OWL tutorial or had an equivalent experience with OWL. The mean self-reported expertise level (Likert-scale, 1 (Novice) - 5 (Expert)) was 2.47 (standard deviation 0.99) for Protégé and 2.53 (standard deviation 1.06) for OWL. Out of the 15 (9 female) participants, there were 3 students, 5 PhD students, 3 research fellows, 1 data scientist, 1 assistant director for information management, 1 clinician, 1 information architect and 1 bioinformatician. An Amazon Voucher (£10) was given to those that were willing to take part.

Experimental Setup. Participants were asked to perform 10 typical ontology authoring tasks in the context of an ontology about pizza (726 axioms, $\mathcal{SHOIN}$) using 5 pre-defined tabs in Protégé. The ontology, like all ontologies used in this work, was chosen to contain mostly "general knowledge" that all participants could relate to. Note that the particular modelling patterns used in an ontology are not pertinent to our problem: A broken subsumption relation or an unsatisfiable class is an indication of erroneous modelling no matter what exact (design) patterns were employed. A task typically involved an action, such as adding a definition and running the reasoner, or an act of exploration, such as inspecting the changes that occurred (i.e. verification), before answering one or more verification questions and finally rating all five tabs for the suitability of performing the task and/or the verification. The five pre-designed tabs were: a simple list of inferred axioms ("Inferences" view in Protégé), the Protégé "Classes" tab, the Inference Inspector, the Protégé "Individuals" tab and the DL Query tab of the DL Query plugin for Protégé. For navigation purposes, all views showed an asserted class hierarchy on the left-hand side, which participants were instructed to ignore when evaluating the suitability of the views for each task. The ten modelling tasks were selected by the authors[11] to cover a wide range of ontology authoring scenarios: (1) under-

[11] The authors have a decade-long combined track record of teach-

standing the topic of the ontology, (2) identifying unsatisfiable classes, (3) repairing unsatisfiable classes, (4) verifying the repair of an unsatisfiable class, (5-6) verifying the definition of a new class, (7) changing the definition of an existing class, (8) verifying the loosening of a restriction by removing a disjointness axiom, (9) verifying the change of an object property assertion and (10) verifying the addition of a property chain (see Appendix Appendix E for a more detailed breakdown). To avoid participant bias, we presented the Inference Inspector simply as a third-party plugin (one of many in the course of the tutorial), rather than our own work. Participants were not formally introduced to the Inference Inspector prior to the experiment, but some of the basic functionality was covered as part of the preceding OWL tutorial. The study participants completed the tasks and questionnaire in parallel. After a maximum duration of 50 minutes, the study was interrupted.

4.2. Results

Figure 2 shows the views preferred by users for tackling a given ontology authoring problem. The participant was allowed to select a single view that was considered the most suitable for addressing a problem. For identifying unsatisfiable classes, at least 5 participants rated the Inference Inspector as the preferred view, compared to 8 who preferred the Classes tab. The Inference Inspector presents unsatisfiable classes clearly to the user, but so does Protégé. The result suggests that at the very least, for this important task, the Inference Inspector is, in fact, usable. That 8 participants preferred the class hierarchy may be a result of familiarity bias — the cognitive bias whereby users may rate experiences as more favourable if they are more familiar. The same argument goes for the repair of unsatisfiable classes, which both views support by allowing the user to delete axioms occurring in a justification. With respect to adding and changing definitions or restrictions, we expected the Inference Inspector to outperform because it makes the changes explicit and not subject to a potentially complicated search in the class hierarchy. When, however, users were asked to explore the consequences of defining a new named class, they preferred the class hierarchy (9 out of 15). This may be because changes with respect to the defined class are made explicit simply by its position in the hierarchy: all sub- and super-classes are new. Only a third of participants preferred the Inference Inspector for this task, possibly because a visual representation of the class hierarchy is easier to understand than a list of axioms. The Inference Inspector did add value, however, when it came to understanding the consequences of a change in the definition (i.e. EquivalentClasses axiom) of an existing class. Seven users preferred the Inference Inspector to six who preferred the class hierarchy, possibly because it is harder to detect a change in the position of a class than it is to see the introduction of a new one. The first prototype of the Inference Inspector did poorly on problems involving individuals, perhaps because the ABox inferences were not ordered or grouped in any way, which was subsequently improved before the later study. Participants of the study were also primed by the ABox heavy (i.e. concerned with modelling instance level knowledge) advanced OWL tutorial, where we made considerable use of the individuals tab, further increasing the potential for familiarity bias.

Figure 3 shows the distribution of scores, on a 5 point Likert scale, the Inference Inspector received for key usability criteria. Ease of use was the weakest point of the Inference Inspector (mean rating of 2.93). While this can be partially attributed to a lack of familiarity with the tool (“it seems useful but I don’t understand it”), participants also found the plugin a “little busy, layout-wise”, “overwhelming to use at first”, with “maybe too much information/options in the same view” (see next paragraph on free-form feedback). As a consequence of this feedback, we reduced the options shown to the user and adopted features such as the axiom renderer from Protégé to ensure a more familiar look and feel before conducting our second user study. Reliability had a mean rating of 3.47, with at least 4 participants giving it a low rating of 2. The problem with reliability may also have resulted from a lack of familiarity — users may not have been sure of what to expect, and were therefore confused by the feedback. Response time was generally rated good (4.27), despite the Inference Inspector being slower than the reasoner in Protégé. It remains to be seen how well the Inference Inspector scales. Users expressed interest in using the Inference Inspector for their own work (3.87) and would recommend it to others (3.87).

Study participants were also asked to provide some free-form feedback. They recognised the importance of prioritising and reducing the information shown (independent of whether the Inference Inspector succeeded at this task): searching the class hierarchy might be “very tedious in a [..] large ontology [..].” and “some means to filter and sort the output” is required, for example by "prioritizing inferences related to unsatisfiability". Moreover, participants recognised the impor-

ing OWL and frequently supervising and engaging in ontology engineering efforts. The tasks were selected based on that experience.

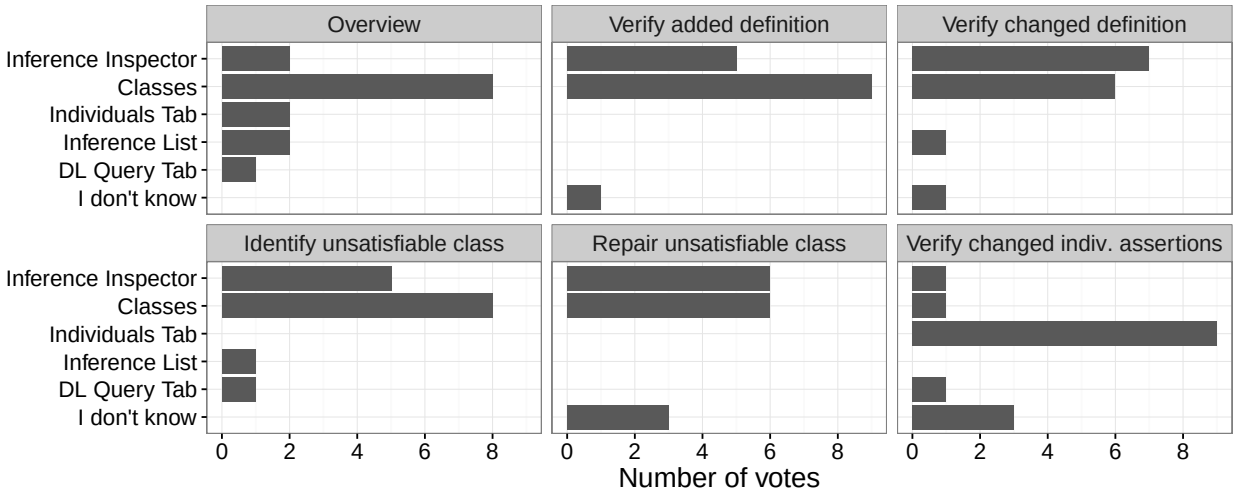


Figure 2: Exploratory study. Participants were asked which was their preferred view for addressing a problem; single vote per participant.

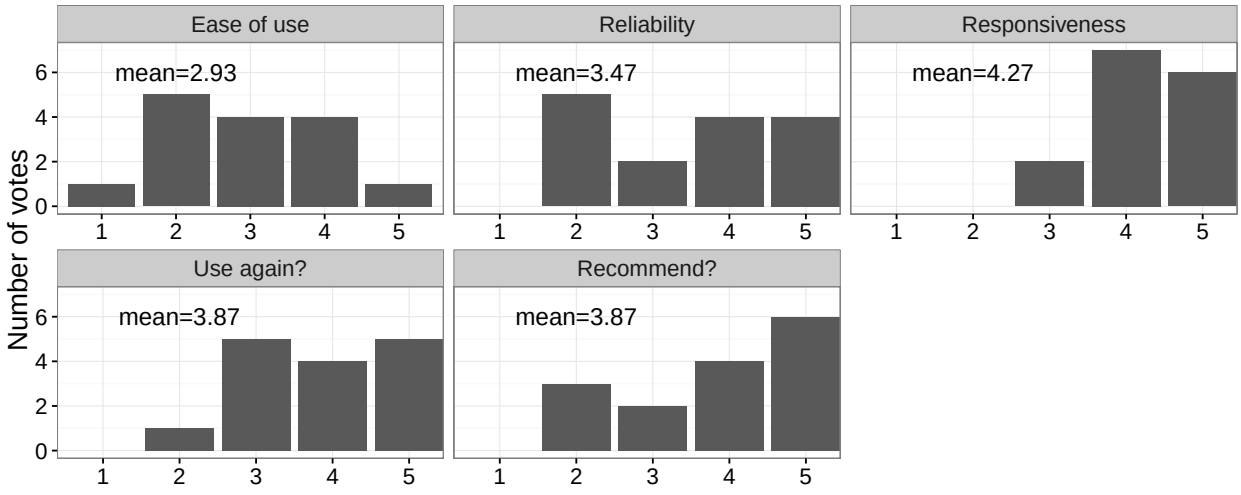


Figure 3: Exploratory study. Ratings of the Inference Inspector (1: unusable, 5: perfectly usable)

tance of immediate feedback: "it'd be helpful to have more indication that the repair was successful. As of now, we're looking for the absence of error warnings. If Protégé could compare the states and flash a message saying 'fixed' [..], it'd be better." This is exactly the sort of feedback we are trying to provide. As the participants were not told about the authors' affiliation with the tool, there was little risk of experimenter bias. Unsurprisingly the feedback on the Inference Inspector was mixed. The Protégé Classes tab was rated as favourite for most (TBox related) tasks: "It's the golden standard for all Protégé views", "I felt this view was the most helpful for editing and [..] information." Participants, however, also recognised the potential impact of familiarity bias: "Maybe I like it because I am familiar with it".

The two main conclusions we draw from this exploratory study are:

- The Inference Inspector approach appears to work best when knowledge over an existing signature was changed (restrictions or definitions on existing classes). The evidence gathered is based on user ratings and free-form feedback, and is therefore indicative, not conclusive. For our formal evaluation, presented in the next section, we will introduce a number of objective performance metrics to confirm our observations.
- The main weakness of the Inference Inspector, according to the feedback and the usability scores, is its ease of use. As a consequence, we simplified the interface considerably before setting up the controlled study. For example, we originally (for terseness) presented the axioms in DL syntax, which we subsequently replaced with the Protégé internal axiom renderer to ensure visual consistency. We furthermore removed unnecessary interface elements (in particular a number of inessential configuration options and checkboxes to switch filters on and off) and replaced text labelled buttons with mnemonic icons to make the Inference Inspector appear less intimidating.

5. Inference Inspector: Formal Evaluation and Hypothesis Test

Our second study was a controlled laboratory experiment designed to assess the extent to which the Inference Inspector helped ontology authors to verify their authoring actions. In particular, we were interested in identifying the groups of verification problems for which the Inference Inspector outperformed Protégé and those where it provides little or no benefit. In addition, we sought to verify the following overarching hypothesis:

Making gained and lost entailments explicit improves the user's understanding of consequences of authoring actions compared to a hierarchy/frame-based view.

5.1. Materials and Methods

To asses the performance of the tools supporting a user at verification, we defined a number of metrics that were based on four measures:

- Answer options: The selected answer options of the verification questions
- Duration: Time to answer the question
- User suitability rating: Likert-scale between 0 (unusable) and 4 (perfectly usable)
- GUI interaction events: Mouse clicks and scroll amount

Since all questions were presented as multiple-choice (i.e. a question with a set of answer options), we were able to distinguish answer options that were (a) correct and selected (true positives), (b) incorrect and selected (false positives, i.e. the participant selected an answer that was wrong), (c) correct and not selected (false negatives, i.e. answers that should have been selected but were missed) and (d) incorrect and not selected (true negatives, i.e. answer that should not have been selected and were not).

Task duration as a performance measure is biased in at least one significant way: if a participant were to randomly select answers as soon as the question appeared, it would seem she succeeded at the task in a small period of time (high performance). We, therefore, defined the following metrics that account for the correctness of the answer:

- Correctness of understanding: #true answer options/#options
- Speed of understanding: time to completion / correctness
- Ease of understanding: #mouse-click / correctness, scroll time / correctness

The "speed" (and ease of understanding) metrics account for answer correctness: the more incorrect the answer, the higher the penalty. For example, if the question was answered correctly (correctness=1.0), then "speed" = "time to completion". The more incorrect the answer, the higher the penalty, with correctness=0 being interpreted as "no understanding reached". In the final analysis, for the purpose of aggregation, we excluded "speed" and "ease of use" measurements where the correctness equalled 0 (4 questions).

Participants. 19 (5 female) participants, between 22 and 57 years of age (mean 33.28), were recruited via word-of-mouth and email advertisement. The background of the participants ranged from MSc students and those with intermediate experience to academics and non-academic professionals with a high level of OWL expertise. The mean self-reported expertise level (Likert-scale, 1 (Novice) - 5 (Expert)) was 3.53 (standard deviation 0.61) for Protégé and 3.68 (standard deviation 0.75) for OWL. Of the 19 participants, 5 were students, 5 PhD students, 5 academics and four non-academics. Four described themselves as ontology engineers, 8 as ontology researchers and 6 as ontology tool developers, and one as 'other'.

Experimental Setup. The controlled study was conducted in a designated usability lab. All participants used the same machine with a 24-inch monitor and Protégé 5.0.0 installed. Tasks were designed for the following verification problems (verification questions): tightening conceptualisation (adding restrictions, verifying consequences), loosening conceptualisation (removing restrictions, fixing unsatisfiable classes) and changing conceptualisation (changing class definitions). In our exploratory study (Section 4), we identified modelling tasks for which the Inference Inspector might be particularly well suited, and this study, therefore, focuses on tasks that change knowledge about entities that are already in the domain, rather than introduce new signature, such as class names. Although this places some limits on the generalisability of the results, we strongly believe that modelling actions affecting existing classes and properties are a considerable, if not the major, fraction of all ontology authoring activities, and the findings are therefore of broad relevance.

To mitigate the impact of varying user expertise levels, all tasks were designed in pairs, i.e. two very similar tasks were designed with one being tested using the Inference Inspector and the other one being tested using the Classes or the Individuals tab. No task required access to the properties tabs. Tabs were assigned to tasks using a Latin square, and then randomly sampled. The survey contained a total of 14 verification tasks. The TBox focused tasks were presented in a scenario involving an ontology about pizza (604 axioms, $\mathcal{SHOIN}$), and the ABox focused tasks were presented in a scenario involving an ontology about family history (89 axioms, $\mathcal{SHIF}$). Participants were asked to answer 2-3 exploration questions for every task, most of which were of the sort "Did the class hierarchy change?" or "Which are the new subclasses of X?". In order to increase participant focus, all questions were auto-submitted after 60 seconds. The questions were designed such that they could be answered comfortably within that time period by a reasonably experienced user of Protégé (see Appendix F for a detailed breakdown of the tasks). Answers submitted due to a timeout were considered to be valid answers in our analysis. This is justifiable because (1) we were explicitly interested in verification performance under a reasonable time constraint and (2) the alternative, discarding such answers, would have deprived us of interesting partial answers, which were quite likely due to the fact that most questions required verifying changed inferences across multiple entities. Within 60 seconds, it could easily be possible that 3 out of 4 entities in a question were correctly verified. Discarding such answers would have led to unnecessary information loss and a potential bias: consider the possibility that all questions verified using Protégé would have taken just above one minute to complete and all partial or almost complete answers would not count at all. The study had a maximum duration of 50 minutes.

Protégé Survey Tool. Although research into human-ontology interaction is increasing, the infrastructure to support data collection still provides only relatively crude change-logs. We previously extended Protégé to allow for the collection of low-level interaction events [39] allowing for richer data to be collected. However, a number of problems remain:

- It is considerably harder to gather answers and measure task completion time when we have to rely on screen casts and external media such as questionnaire software (see exploratory study, Section 4). We needed an easier and more reliable way to collect answers and measure task duration.
- We felt that supplying the tasks and questions as separate documents put an unnecessary cognitive strain on the participant (switching back and forth).
- We wished to collect eye tracking data; it is known that frequently looking down on a piece of paper with instructions for example significantly hampers the possibility of collecting gaze data (espe-

cially if looking down comes with shifts in the participant's overall posture).

- In our exploratory study (Section 4) we asked the participants, for each verification question, to evaluate *all* views, i.e. to check all available Protégé views including the Inference Inspector for how suitable they were to support answering the question. This approach is heavily biased by the order in which the participant chooses to evaluate the views, and doesn't allow the introduction of meaningful objective performance measures such as question-answer duration or answer correctness. To measure the effectiveness of the views for verification, random assignments of views to problems are preferable. We needed a system that allowed us to randomly sample assignment of views to verification problems and only show the participants the one view we wanted them to evaluate.

We therefore designed and implemented the Protégé Survey Tool[12] (PST). The PST is implemented as a Protégé plugin and allows setting up or simulating ontology authoring tasks such that can be evaluated by the user and can, therefore, be used for setting up usability studies and A/B tests. The PST gives access to data that are seldom reported in the literature, such as precise task duration, click count, scroll time, key presses and details of answer accuracy. We believe that this tool can help the ontology authoring community to design more reliable user studies with richer analyses.

Each PST session is configured with a survey. A survey has three main components: a set of scenarios, a set of tasks (for each scenario) and a set of questions (for each task). A scenario is associated with an ontology; when the PST runs a new scenario, it will load an ontology (from the web or locally) and present some description to the study participant such as: "You are an ontology engineer hired by Pizza Hug. Your boss charged you with changing the ontology according to new company policies." For each scenario, the participant is presented with a set of tasks, such as "You decide that it would be best to define Vegetarian Pizza as Pizza that hasIngredient only VegetarianIngredient." These tasks can either be completed by the participant or simulated by the PST (i.e. it will automatically add the right axiom). In either case, the PST will check whether the task was completed correctly before continuing. Each task can be associated with a reasoning request, which means that the task is not finished until the reasoner has been run. For each such task, the PST presents a number of questions, for example, verification questions of the form "Is Margherita correctly inferred to be vegetarian?". At the moment, multiple choice, single choice and text questions are available as question types. The PST automatically gathers timings for all questions. It is possible to specify a timeout, after which the question will be auto-submitted. Each question can be associated with a Protégé tab (custom or built-in), such as the Inference Inspector tab or the Classes tab, in our case. The surveys themselves are implemented as XML documents. The PST gathers data such as false positives and negatives and true positives and negatives from the answer set, time to completion, click count, key strokes and scroll time. Protégé was pre-configured with the Inference Inspector and the PST to administer the survey. The PST can be obtained from the supplementary material website.

[12] `https://github.com/matentzn/protegesurvey`

5.2. Results

We tested the potential for improvement of verification performance when using the Inference Inspector with respect to (1) a particular range of tasks and (2) the consequences of a change.

While we generalise our results for tasks of the kind described in Section 5.1, we do not say anything about other kinds of modelling tasks, such as those involving datatypes. Secondly, we are interested in understanding the consequences of a change. This means we do not measure the performance of exploration tasks such as "What are the super-classes of *A*" or "Which one of the following are not sub-classes of *A*", but instead "Which are the new sub-classes of *A*" and "Which subclasses were lost to *A*". The difference is subtle, but important. We use a Wilcoxon signed-rank test to determine whether the difference between the Inference Inspector and the respective Protégé views is statistically significant (at a significance level of p=0.05) for a particular metric.

5.2.1. Verification performance

Verification tasks were more likely to be performed correctly with the Inference Inspector (86%), than with the Protégé views (70%, p=0.009), see Table 1. This provides evidence that our hypothesis, for the specified set of tasks, holds. At a closer look (Figure 4) we can see that while the problems solved with the Inference Inspector are clustered at the high end of correctness, the ones solved with Protégé are more evenly spread. The same can be observed for the user ratings. We acknowledge, however, that subjective ratings are potentially unreliable due to experimenter bias.

Tasks were also performed faster with the Inference Inspector; in the case of exploration tasks, the difference was more than 4.7 sec (mean). However, this difference is not statistically significant (p=0.101). Looking at the cases that timed out (i.e. were not completed after 60 seconds), we find that 14.98% of the total number of questions across participants (37 questions) timed out when using the Inference Inspector, compared to 19.03% (47 questions) when using the default Protégé views. If answer correctness is taken into account ("speed", see Section 5.1), the difference in the time for completing questions between the two views is even greater (and significant, p=0.017). Figure 4 shows how the distribution of task performance time (speed) with the Inference Inspector is shifted to the left. The distribution of the task duration, in particular the high density of short-duration tasks, can be explained by the immediacy with which questions such as "Which classes are unsatisfiable?" or "Did the class hierarchy change?" can be answered with the Inference Inspector.

Figure 4 shows how the level to which users needed to scroll is distributed. While there are more tasks that require very little scrolling when using the Inference Inspector, there are also some tasks that require a lot — for some problems, the Inference Inspector shows the results immediately, for others, searching is required. The difference (in terms of scroll effort and correctness) between the Inference Inspector and Protégé however is not statistically significant (p=0.155). The primary form of navigation in the Protégé hierarchy is expanding the nodes rather than scrolling, which manifests itself typically as mouse-clicks. There were considerably fewer clicks involved in arriving at a correct answer using the Inference Inspector (p=0.015).

Looking at false positives and false negatives gives a more fine-grained picture of correctness. False positives are incorrect observations, i.e. question options that were false, but were selected by the user. False negative answers are missed answers, that suggest that the view gave the participant an incomplete picture of the consequences. At first glance, it is surprising that the exploration tasks resulted in more false positive explorations than false negative ones (the converse is true for the Inference Inspector). However, this can be explained by a significant number of (Yes/No/Not sure) questions that were frequently answered incorrectly in the case of the Protégé views, for example "Did the class hierarchy change?", which is not always obvious when using the "Classes" tab.

We discuss the type of verification questions as indicated by the correctness values in Table 2:

- **Tightened restrictions**. With a mean increase of 42.27% in correctness, the effect of the Inference Inspector for verifying tightened restrictions is considerable. A tightened restriction reduces the extension of a class or relation (see first paragraph of section 3), for example by adding a more specific domain restriction to a property or adding a super-class restriction to a class. The consequences of such changes can be hard to track in an SHFB-like view: For example, adding a domain restriction to an object property R can lead to new superclasses for a class A, if A holds an existential restriction of the form $SubClassOf(A, R \text{ some } X)$. Identifying these classes manually in a large ontology is often out of the question.
- **Identifying source of unsatisfiability**. The extent to which the Inference Inspector outperforms Protégé in terms of finding the source of an unsatisfiability is somewhat unexpected, as both approaches support checking for justifications. One possible explanation is that justifications can be found in the Inference Inspector with ease, while Protégé users must first select the unsatisfiable class in the class hierarchy, and then and search for the little question mark next to the inferred super-class "owl:Nothing". The Inference Inspector initially shows only the smallest justification, while Protégé shows all possible ones immediately. That can make finding particular axioms in a justification more difficult.
- **Loosened restriction**. With 27.33% more correct results, the Inference Inspector also outperforms Protégé in terms of verifying loosened restrictions. The rationale is similar to the one for tightened restrictions.
- **Assertion**. Due to their potentially large number, we expected the Inference Inspector to be less well suited to support the verification of ABox related inferences (both FHKB question groups), but at least for class assertions—verifying that changes to a definition resulted in new type inferences for individuals—we were wrong. With 26.07%, the results are again encouraging.
- **Changed definition**. It was surprising to us that changed definitions appeared relatively low in the list; our initial prediction was that this would be the clearest case for the Inference Inspector. However, with 18.86% improved correctness, the results are still very good. Moreover, verifying changed definitions is local to a known class, so finding it in the

	mean		*median*		*sd*		*p*
Metric	II	P	II	P	II	P	
Correctness***	0.86	0.70	1.00	0.75	0.24	0.30	0.009
Rating***	3.51	1.93	4.00	2.00	0.82	1.41	0.003
Duration	31.89	36.66	26.17	33.30	20.73	22.93	0.101
Speed**	46.36	63.93	30.49	46.74	47.22	57.97	0.017
Ease** (mouse-click)	7.27	11.96	4.00	8.50	10.82	12.07	0.015
Ease (scroll-amount)	3.05	6.19	0.00	0.00	14.21	20.05	0.155
False negatives***	0.04	0.10	0.00	0.00	0.21	0.29	0.002
False positives***	0.05	0.25	0.00	0.00	0.22	0.43	0.002
True negatives***	0.79	0.56	1.00	1.00	0.41	0.50	0.002
True positives	0.95	0.73	1.00	1.00	0.22	0.44	0.141

Table 1: Mean, median and standard deviations for key metrics. II is the Inference Inspector, P is Protégé (depending on the task, either individual or Classes tab). P-values larger than 0.05 indicate that differences are not statistically significant. Asterisk next to metric label indicates significance degree: ***:p<0.01, **:p<0.05, *:p<0.1, no star:p>=0.1

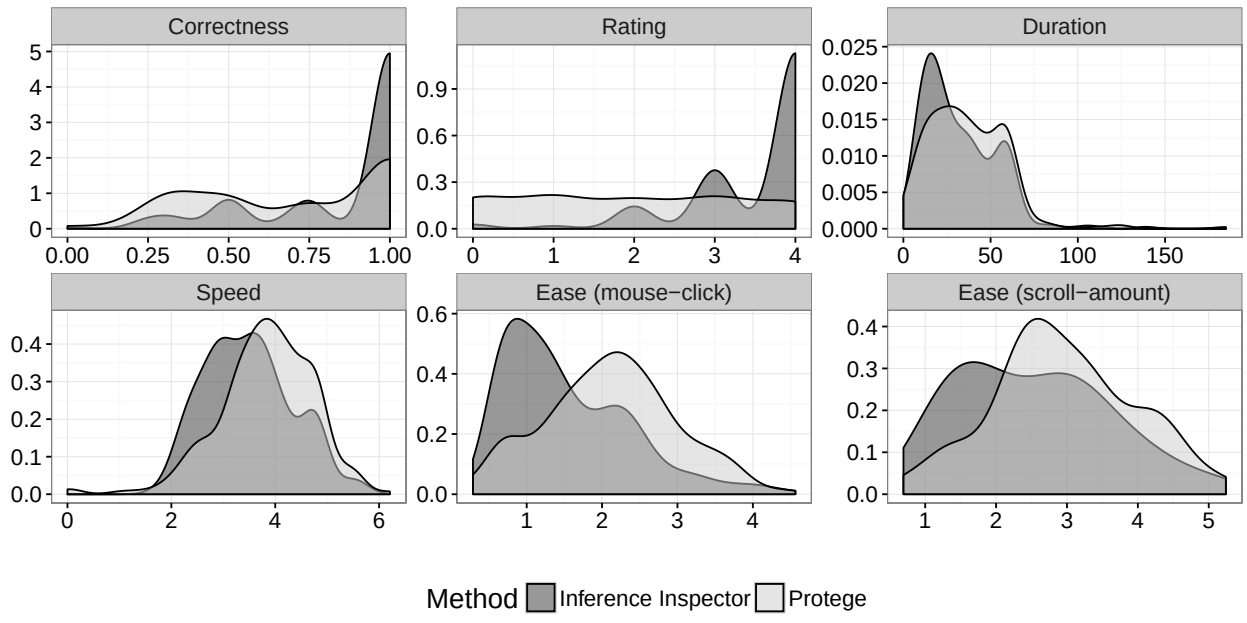


Figure 4: Main study. Kernel density plots for 6 key metrics (x: metric, y: density). Lower three log-rescaled.

class hierarchy is perhaps easier than we thought, while the consequences of loosening or tightening restrictions can potentially affect unexpected entities in the ontology.

- **Fixed unsatisfiability**. The verification as to whether a change to the ontology fixed an unsatisfiable class works quite well in both views, which is unsurprising (given the prominent red rendering of unsatisfiable classes in both views).

The time it takes to answer verification questions using the Inference Inspector negatively correlates with question order (Spearman's rank correlation rho=-0.63, p=0.006), i.e. questions that were asked later were answered faster. This may indicate a significant learning effect, which makes sense given the complete unfamiliarity of all participants with the tool prior to the study. The scenarios and tasks we evaluated in the study told a story of successive changes to the ontology, which precluded any form of randomisation. Therefore, it is at least theoretically possible that this learning effect was due to constantly decreasing difficulty of questions. We cannot, however, observe a significant learning effect when using Protégé (Spearman's rank correlation rho=-0.26, p=0.298), presumably because all participants were familiar with Protégé's default views prior to the study. There is, therefore, a strong possibility that a lack of familiarity masked the extent of the benefits of using the Inference Inspector, and that the positive effect on verification performance might increase as people continue to use the tool.

5.2.2. Post-session Ratings

The distribution of post-session ratings for ease of use, responsiveness and reliability can be seen in Figure 5. Similar to our observations in the exploratory study, ease of use continues to be the lowest rated factor. This effect may be due to a lack of familiarity with the tool,[13] while most participants reported a high level of expertise with Protégé (and therefore its default views). However, the results are encouraging compared to the ratings from the exploratory study. We are aware that the validity of post-session questionnaires, in general, is considerably lower than objective performance measures such as the ones presented in the previous section. We do, however, feel that they are at the very least indicative of a trend, as 8 out of 19 participants declared not being aware who the developers of the Inference Inspector were, which helped to reduce experimenter bias.[14] While the increase in ratings compared to the exploratory study could be partially attributed to increased experimenter bias (during the exploratory study, no participant was aware of the experimenters' affiliation with the tool), we believe that the significantly increased average knowledge of the participants (including OWL professionals, lecturers and professors) further contributed to the increased perceived utility of the Inference Inspector. Lastly, we like to believe that our efforts to simplify the UI based on the feedback of the exploratory study had a positive effect as well. When asked whether they would use the tool in the future, 16 said "yes", 2 said "maybe a better version" and one did not respond. When asked whether they would recommend the tool to others, 17 said "yes", 1 said "maybe a better version" and 1 did not comment.

6. Conclusions

Ontologies can be complex systems of axioms, and a modelling action, such as the addition of an axiom, may have consequences throughout the whole system. Being able to apprehend such consequences is important in ontology authoring in order to prevent both erroneous and unintended modelling. We presented the Inference Inspector — a tool that shows the consequences of modelling actions — and explored the hypothesis that making changes to key entailment sets explicit improves understanding of the consequences of modelling actions.

We find that making entailment set changes explicit improves understanding of the consequences of a range of typical modelling actions. We provide evidence that the standard static view of an ontology does not adequately support people in understanding the consequences of modelling actions, and should be addressed by current ontology authoring environments. Making the consequences of changes explicit as changes to entailment sets is by no means the only, or necessarily best, way to approach this issue. For example, if the set of consequences is restricted to atomic subsumptions, approaches can be (and have been) considered that highlight classes in the class hierarchy whose sub- or superclasses have changed. We hope, however, that our work shows that making changes explicit is a key feature missing from ontology authoring environments based on the static hierarchy/frame-based paradigm and that the Inference Inspector will help ontology authors to verify their modelling choices more easily, thereby improving the ontology authoring experience.

[13] None of the participants had seen it prior to this study.

[14] This was asked explicitly as part of the post-session questionnaire.

Type of Verification Problem	*COR*	*II*	*PRO*	*PCH*
Tightened restriction (Pizza)	0.77	0.91	0.64	42.27%
Unsatisfiablity source (Pizza)	0.58	0.67	0.49	37.84%
Loosened restriction (Pizza)	0.86	0.96	0.75	27.33%
Assertion (FHKB)	0.69	0.77	0.61	26.07%
Changed definition (Pizza)	0.84	0.91	0.77	18.86%
Changed property axioms (FHKB)	0.74	0.76	0.71	7.41%
Fixed unsatisfiability (Pizza)	0.94	0.95	0.93	1.89%

Table 2: Correctness grouped by type of verification question. COR is the overall mean correctness, and indicator of question difficulty. II is the mean correctness value for the Inference Inspector, PRO for Protégé and PCH the percentage change between the two. Sorted by PCH.

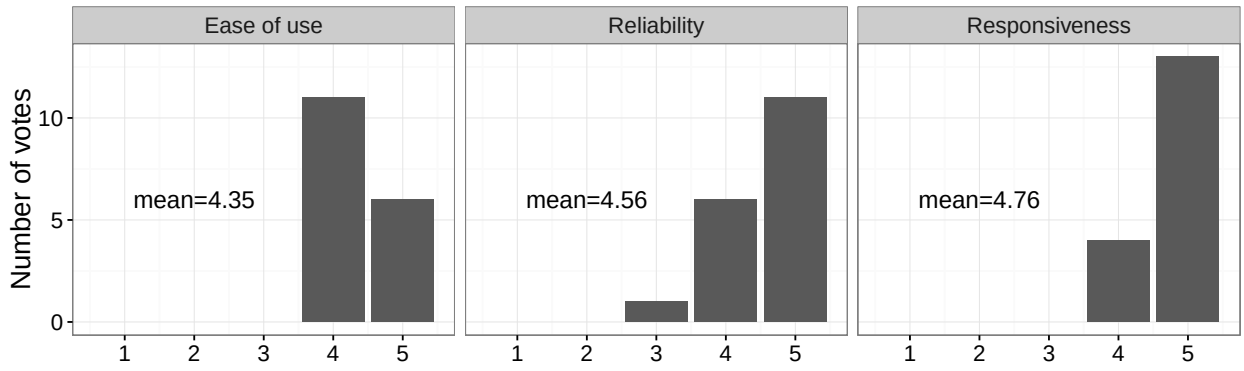


Figure 5: Main study. Ratings of the Inference Inspector (1: unusable, 5: perfectly usable).

The plugin is maintained and available at `https://github.com/matentzn/inference-inspector`. We welcome bug reports and feature requests to be submitted through GitHub's issue tracking system. A demonstration video, along with links to the source code, the tutorial and the results and documentation of both studies, can be found at `http://ow.ly/pK8P300x9wq`.

Acknowledgments: This research has been funded by the EPSRC project WhatIf: Answering "What if..." questions for Ontology Authoring, EPSRC reference EP/J014176/1.

References

[1] S. Bail, B. Parsia, and U. Sattler. Extracting Finite Sets of Entailments from OWL Ontologies. In *Informal Proceedings of the 24th International Workshop on Description Logics (DL-2011), Barcelona, Spain, July 13-16, 2011.*, 2011.

[2] T. G. O. Consortium. Gene Ontology Annotations and Resources. *Nucleic Acids Research*, 41(D1):D530–D535, 2013.

[3] M. Copeland, R. S. Gonçalves, B. Parsia, U. Sattler, and R. Stevens. Finding fault: detecting issues in a versioned ontology. In *Proceedings of the Second International Workshop on Debugging Ontologies and Ontology Mappings, Montpellier, France, May 27, 2013*, pages 9–20, 2013.

[4] R. Denaux, D. Thakker, V. Dimitrova, and A. G. Cohn. Interactive Semantic Feedback for Intuitive Ontology Authoring. In *Formal Ontology in Information Systems - Proceedings of the Seventh International Conference, FOIS 2012, Gray, Austra, July 24-27, 2012*, pages 160–173, 2012.

[5] I. Distinto, M. d'Aquin, and E. Motta. LOTED2: An ontology of European public procurement notices. *Semantic Web*, 7(3):267–293, 2016.

[6] M. Dzbor, E. Motta, C. Buil, J. M. Gomez, O. Görlitz, and H. Lewen. Developing Ontologies in OWL: an Observational Study. In *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11, 2006*, 2006.

[7] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[8] C. Golbreich, M. Horridge, I. Horrocks, B. Motik, and R. Shearer. OBO and OWL: Leveraging Semantic Web Technologies for the Life Sciences. In *SEMWEB*, 2007.

[9] R. S. Gonçalves, N. Matentzoglu, B. Parsia, and U. Sattler. The Empirical Robustness of Description Logic Classification. In *Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013*, pages 277–280, 2013.

[10] P. Haase, F. v. Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure. A Framework for Handling Inconsistency in Changing Ontologies. In *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings*, pages 353–367, 2005.

[11] M. Horridge. *Justification-based Explanation in Ontologies*. PhD thesis, University of Manchester, 2011.

[12] M. Horridge, S. Bail, B. Parsia, and U. Sattler. The Cognitive Complexity of OWL Justifications. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pages 241–256, 2011.

[13] V. Ivanova and P. Lambrix. A Unified Approach for Aligning Taxonomies and Debugging Taxonomies and Their Alignments. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pages 1–15, 2013.

[14] K. Kaljurand. ACE View — an Ontology and Rule Editor based on Attempto Controlled English. In *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008*, 2008.

[15] A. Kalyanpur, B. Parsia, E. Sirin, and J. A. Hendler. Debugging unsatisfiable classes in OWL ontologies. *J. Web Sem.*, 3:268–293, 2005.

[16] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. G. Giannopoulou. Ontology visualization methods - a survey. *ACM Comput. Surv.*, 39, 2007.

[17] Y. Kazakov. RIQ and SROIQ Are Harder than SHOIQ. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR-2008), Sydney, Australia, September 16-19, 2008.*, pages 274–284, 2008.

[18] Y. Kazakov, M. Krötzsch, and F. Simancik. The Incredible ELK - From Polynomial Procedures to Efficient Reasoning with EL Ontologies. *Journal of Automated Reasoning*, 53(1):1–61, 2014.

[19] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *SEMWEB*, 2004.

[20] P. Lambrix, M. Habbouche, and M. Pérez. Evaluation of ontology development tools for bioinformatics. *Bioinformatics*, 19(12):1564–1571, 2003.

[21] M. Lee, N. Matentzoglu, B. Parsia, and U. Sattler. A Multi-reasoner, Justification-Based Approach to Reasoner Correctness. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, pages 393–408, 2015.

[22] S. F. Liang, R. Stevens, D. Scott, and A. L. Rector. Automatic Verbalisation of SNOMED Classes Using OntoVerbal. In *Artificial Intelligence in Medicine - 13th Conference on Artificial Intelligence in Medicine, AIME 2011, Bled, Slovenia, July 2-6, 2011. Proceedings*, pages 338–342, 2011.

[23] N. Matentzoglu and B. Parsia. The OWL Full/DL Gap in the Field. In *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED 2014) co-located with 13th International Semantic Web Conference on (ISWC 2014), Riva del Garda, Italy, October 17-18, 2014.*, pages 49–60, 2014.

[24] N. Matentzoglu and B. Parsia. *BioPortal Snapshot 27.01.2015*. Feb. 2015.

[25] D. L. McGuinness and P. F. Patel-Schneider. Usability Issues in Knowledge Representation Systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 608–614, 1998.

[26] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles (Second Edition), Dec. 2012.

[27] C. Mungall, H. Dietze, and D. Osumi-Sutherland. Use of OWL within the Gene Ontology. In *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED 2014) co-located with 13th International Semantic Web Conference on (ISWC 2014), Riva del Garda, Italy, October 17-18, 2014.*, pages 25–36, 2014.

[28] T. A. T. Nguyen, R. Power, P. Piwek, and S. Williams. Measuring the Understandability of Deduction Rules for OWL. In *Proceedings of the First International Workshop on Debugging*

Ontologies and Ontology Mappings, WoDOOM 2012, Galway, Ireland, October 8, 2012., pages 1–12, 2012.

[29] N. F. Noy, N. H. Shah, P. L. Whetzel, B. Dai, M. Dorf, N. Griffith, C. Jonquet, D. L. Rubin, M.-A. D. Storey, C. G. Chute, and M. A. Musen. BioPortal: Ontologies and Integrated Data Resources at the Click of a Mouse. *Nucleic Acids Research*, 37(Web-Server-Issue):170–173, 2009.

[30] B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, and A. Steigmiller. The OWL Reasoner Evaluation (ORE) 2015 Competition Report. In *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS-2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, 2015.

[31] B. Parsia, E. Sirin, and A. Kalyanpur. Debugging OWL ontologies. In *WWW*, 2005.

[32] A. Parvizi, C. Mellish, K. v. Deemter, Y. Ren, and J. Z. Pan. Selecting Ontology Entailments for Presentation to Users. In *KEOD 2014 - Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Rome, Italy, 21-24 October, 2014*, pages 382–387, 2014.

[33] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa. OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. *Int. J. Semantic Web Inf. Syst.*, 10(2):7–34, 2014.

[34] A. L. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In *EKAW*, 2004.

[35] R. L. Richesson, J. E. Andrews, and J. P. Krischer. Use of SNOMED CT to Represent Clinical Research Data: A Semantic Characterization of Data Items on Case Report Forms in Vasculitis Research. *Journal of the American Medical Informatics Association*, 13(5):536–546, 2006.

[36] T. Tudorache, C. I. Nyulas, N. F. Noy, and M. A. Musen. WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the Web. *Semantic Web*, 4:89–99, 2013.

[37] M. Vigo, S. Bail, C. Jay, and R. D. Stevens. Overcoming the pitfalls of ontology authoring: Strategies and implications for tool design. *Int. J. Hum.-Comput. Stud.*, 72(12):835–845, 2014.

[38] M. Vigo, C. Jay, and R. Stevens. Design insights for the next wave ontology authoring tools. In *CHI Conference on Human Factors in Computing Systems, CHI'14, Toronto, ON, Canada - April 26 - May 01, 2014*, pages 1555–1558, 2014.

[39] M. Vigo, C. Jay, and R. Stevens. Protégé4us: Harvesting Ontology Authoring Data with Protégé. In *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, pages 86–99, 2014.

[40] M. Vigo, C. Jay, and R. Stevens. Constructing Conceptual Knowledge Artefacts: Activity Patterns in the Ontology Authoring Process. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*, pages 3385–3394, 2015.

[41] H. Wang, T. Tudorache, D. Dou, N. F. Noy, and M. A. Musen. Analysis of User Editing Patterns in Ontology Development Projects. In *OTM 2013 Conferences*, pages 470–487. Springer Berlin Heidelberg, Sept. 2013.

[42] T. D. Wang and B. Parsia. CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, pages 695–708, 2006.

[43] P. Warren, P. Mulholland, T. D. Collins, and E. Motta. The Usability of Description Logics - Understanding the Cognitive Difficulties Presented by Description Logics. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, pages 550–564, 2014.

Appendix A. Entailment Set Definitions

We call $\widetilde{O}$ the signature of the ontology (the set of all named entities in O), with $\widetilde{O}_P \subseteq \widetilde{O}$ being the set of all object property names in O, $\widetilde{O}_C \subseteq \widetilde{O}$ being the set of all class names, $\widetilde{O}_I \subseteq \widetilde{O}$ being the set of all individual names and $\widetilde{O}_D \subseteq \widetilde{O}$ being the set of all data property names. Let us consider the set of all OWL 2 entailments. The set of all OWL 2 entailments is, for a given ontology O, the set of all legal OWL 2 DL axioms according to the specification [26], excluding those violating global constraints, [15] that follow from O. This set is of infinite cardinality: consider the axioms $SubClassOf(A, R \text{ some } B)$, SubClassOf($A$, R some (R some B)), SubClassOf(A, R some (R some (R some B))) and so on. We are interested in the following *finite* entailment sets [1].

Informally, the set of atomic subsumptions corresponds to the set of all axioms of the form SubClassOf(A, B) that follow from the ontology, where A and B are named classes. We can formally define this as follows: Given an ontology O, the set of atomic class subsumptions is the set of all axioms of the form SubClassOf(A, B) for all $A, B \in \{\widetilde{O}_c \cup \top \cup \bot\}$ with $O \models$ SubClassOf(A, B) [16]. The worst case size of the entailment set is n^2, where n is the number of class names in $\widetilde{O}$. Atomic object property subsumptions and atomic data property subsumptions can be defined respectively.

A property characteristic is an axiom of the sort $x(P)$, where P is an object property expression, [17] e.g. FunctionalObjectProperty(P) or ReflexiveObjectProperty(P). Given an ontology O, the set of all object properties in the signature of O, $\widetilde{O}_P$ and the set of all legal OWL 2 object property characteristics $\mathfrak{X}$, the *atomic property characteristics* entailment set is the set of all axioms of the form $x(P)$ where $x \in \mathfrak{X}$ and $P \in \widetilde{O}_P$ with $O \models x(P)$. The worst case size of the entailment set is $n * 7$, where n is the number of object property names in $\widetilde{O}$ and 7 is the number of legal property characteristics in OWL 2.

[15] `https://www.w3.org/TR/owl2-syntax/\#Global_Restrictions_on_Axioms_in_OWL_2_DL`

[16] Note that there are other ways of defining it: E.g. without $\top$ or $\bot$.

[17] `https://www.w3.org/TR/owl2-syntax/\#Object_Property_Axioms`, Fig. 15

For brevity, we will enumerate the remaining entailment sets in the following. Given an ontology $\mathcal{O}$, the set of

- *equivalent classes* is the set of all axioms of the form EquivalentClasses(A, B) for all $A, B \in \{\widetilde{\mathcal{O}_C} \cup \top \cup \bot\}$ with $\mathcal{O} \models$ EquivalentClasses(A, B). Worst case size of the entailment set is n^2, where n is the number of class names in $\widetilde{\mathcal{O}}$. Object property equivalences and data property equivalences are defined respectively.
- *disjoint classes* is the set of all axioms of the form DisjointClasses(A, B) for all $A, B \in \{\widetilde{\mathcal{O}_C} \cup \top \cup \bot\}$ with $\mathcal{O} \models$ DisjointClasses(A, B). Worst case size of the entailment set is n^2, where n is the number of class names in $\widetilde{\mathcal{O}}$.
- *class assertions* is the set of all axioms of the form ClassAssertion(a, B) for all $a \in \widetilde{\mathcal{O}_I}, B \in \{\widetilde{\mathcal{O}_C} \cup \top\}$ with $\mathcal{O} \models$ ClassAssertion(a, B). Worst case size of the entailment set is n^m, where n is the number of class names in $\widetilde{\mathcal{O}}$ and m is the number of individual names in $\widetilde{\mathcal{O}}$.
- *object property assertions* is the set of all axioms of the form ObjectPropertyAssertion(a, b, R) for all $a, b \in \widetilde{\mathcal{O}_I}, R \in \{\widetilde{\mathcal{O}_P} \cup \top\}$ with $\mathcal{O} \models$ ObjectPropertyAssertion(a, b, R). Worst case size of the entailment set is n^2, where n is the number of individual names in $\widetilde{\mathcal{O}}$.

Appendix B. Other Inference Inspector Features

Apart from entailment set changes, the Inference Inspector analyses the added or changed axioms themselves, and determines whether the axiom (1) violates any of the OWL 2 profiles (DL, EL, QL, RL), (2) introduces an entity, such as a class name, that is not mentioned in any other axiom in the ontology (stray signature), (3) was implied before it was added (redundancy) and (4) is a tautology.

Profile violations are produced when an axiom does not adhere to the profile's specification [26]. For example, the use of universal restrictions (OWLObjectAllValuesFrom) is not allowed in OWL 2 EL; an axiom using a universal restriction would, therefore, be marked as violating the EL profile specification. The main motivation for presenting profile violations is that some ontologies are built with applications in mind that need reasoning at runtime. These applications may want to use an efficient specialised OWL 2 EL reasoner such as ELK [18]. Ontology authors may, therefore, benefit from a warning when adding an axiom that takes the ontology outside of OWL 2 EL. In the context of the Inference Inspector, profile violations are determined using the profile checking facilities of the OWL API. This is not always ideal, but as far as we know, it is the best tool that currently exists for the purpose. A detailed breakdown of OWL 2 profile violations across a wide range of popular ontologies can be found in [23].

A tautology is an axiom that is always true, for example, SubClassOf$(A, Thing)$. There are few cases where it is necessary to add a tautology explicitly to the ontology. Therefore, the Inference Inspector tells the user if an added axiom is, in fact, a tautology, which is determined by checking whether the axiom is entailed by the empty ontology.

Stray signature is created if an axiom is added that mentions a named entity (class, property, individual) that is never mentioned together with another named entity in the ontology. These kinds of axioms are typically tautologies like SubClassOf(A,Thing) or ClassAssertion(a,Thing). An example of a non-tautological axiom that would be classified as stray signature is SubClassOf$(A, \neg A)$, given that A does not appear in any other axiom in the ontology. The motivation of notifying the user that a stray axiom was added lies in the possibility of (unintentionally) using a wrong name when adding an axiom.

Lastly, ontology authors may be interested in learning that an axiom α just added is already implied by $\mathcal{O} \setminus \{\alpha\}$, i.e. is redundant. As determining this at runtime may be very costly, the feature is deactivated by default but can be switched on by the user. This feature is implemented again exploiting justifications. First, two justifications are generated. If at least one of the justifications does not contain the axiom, we know that it is implied by $\mathcal{O} \setminus \{\alpha\}$. This works, because of the following. Given an axiom α, an ontology $\mathcal{O}$ with $\alpha \in \mathcal{O}$ and $\mathfrak{J}$ the set of all (unique) justifications for α, the following holds: $\mathcal{O} \setminus \{\alpha\} \models \alpha \Leftrightarrow |\mathfrak{J}| >= 2$. First the $\Leftarrow$ direction: if $|\mathfrak{J}| >= 2$ then there must be at least one justification that does not contain α, so $\mathcal{O} \setminus \{\alpha\} \models \alpha$ follows immediately. This is because a justification that contains α must be equal to $\{\alpha\}$ (a so-called self-justification); if it contained more axioms, it would not be minimal, which justifications are by definition. Next the $\Rightarrow$ direction: if $\mathcal{O} \setminus \{\alpha\} \models \alpha$, there must be at least two justifications: the self-justification and at least another one that does not involve α.

Appendix B.1. Restrictions for computing entailment sets

In order for the Inference Inspector to work correctly, the reasoner used for computing the entailment sets needs to be implement the following (`OWLReasoner`) interface methods correctly: getTypes(OWLNamedIndividual i,boolean direct), getTypes(e,direct), isEntailed(ax), getDisjointClasses(e), getEquivalentClasses(e), getEquivalentDataProperties(d), getEquivalentObjectProperties(p), getObjectPropertyValues(e, i), getSuperClasses(c, direct), getSuperObjectProperties(c, direct) and getSuperDataProperties(p, direct). For ontologies of reasonable complexity and size, we recommend HermiT [7]. However, the selection of the reasoner is up to the user.

Appendix C. Priorities in the Inference Inspector

List of priorities and their codes as implemented by the Inference Inspector.

- P1: Inferences of critical importance. By default, these are ontology inconsistency and class unsatisfiability. P1 inferences are highlighted in red and are only removed from the view when they are not anymore entailed.
- P2: Important inferences. By default these are equivalent classes, subclass relationships that do not correspond to equivalent classes (SubClassOf(A, B) where EquivalentClasses(A, B) does not hold), direct disjointness (no superclasses of the disjoint classes are disjoint), most specific type (ClassAssertion(a, B) where there is no ClassAssertion(a, C) with SubClassOf(C, B)) and OWL 2 DL profile violations.
- P3: Inferences of medium importance. This is the default priority.
- P4: Inferences that are potentially less interesting. By default, these are asserted axioms (inferences that are asserted), property characteristics, axioms of the form EquivalentObjectProperty(R,inverse(S)), indirect class disjointnesses, subclass relationships that do not correspond to equivalent classes (SubClassOf(A, B) where EquivalentClasses(A, B) holds), other profile violations and tautologies.
- REMOVE: Inferences that are of no interest to the user and will be excluded from the view. By default, no inferences are removed from view. However, to save computational costs, determining whether an added axiom is redundant is deactivated by default.

Appendix D. BioPortal Survey Setup

The BioPortal survey was conducted to determine the distribution of axiom types across ontologies on the web, in particular to identify axiom types *of general interest*, i.e. that are used (frequently) across ontologies. While BioPortal does not encompass all ontologies that have been made available on the web, it serves as a proxy that is popular for ontology surveys and reasoner benchmarking [30]; it is reasonably large (more than 300 ontologies) and it is maintained by active communities of ontology developers. We used a BioPortal snapshot from September 2015 [24].

From each ontology in BioPortal that was parseable with the OWL API, and each axiom in this ontology, we extracted features such as its type (SubClassOf, ClassAssertions, etc) and counts of the types of its nested class expressions. For example {OWLClass,2} {OWLObjectSomeValuesFrom,1} means that a particular axiom contained two (not necessarily unique) OWL class names and one existential restriction. We assumed that the distribution of axiom types from the set of asserted axioms is somehow similar to the distribution from the set of inferred axioms (given our entailment sets). This is reasonable, at least for ontologies that involve mainly hierarchical modelling on the TBox level (i.e. atomic subsumptions). By determining the distribution of the asserted axiom types, we aimed to show that we cover a significant proportion of relevant entailment sets when it comes to ontologies on the web.

Appendix E. Tasks exploratory study

- Task 1: Understanding the ontology. Open the ontology pizza.owl located at X and run the reasoner. Get a sense of the ontology. What is it about? What are the main classes? What seem to be the dominating inferrences of interest?
- Task 2: Unsatisfiable classes. Find out whether there are any unsatisfiable classes in the ontology (excluding the built-in class owl:Nothing!). How many unsatisfiable classes does the ontology have?
- Task 3: Repairing unsatisfiable classes. See whether you can find a quick way to repair the unsatisfiable classes (use any one view that you like). Only remove restrictions (SubclassOf) or definitions (Equivalent to). DO NOT RUN THE REA-

SONER YET. Which tab(s) did you use to (try to) repair the unsatisfiable classes?

- Task 4: Verifying the repair. Run the reasoner only once (just after you performed your repair). Note that it does not matter whether you actually succeeded in performing the repair or not. Did you successfully repair the unsatisfiable classes?
- Task 5: Defining VegetarianPizzaAlternative1. Click on "Thing" in the class hierarchy (exp2). Create a new class VegetarianPizzaAlternative1 (use exp2). Define it as "Pizza and (hasTopping only VegetarianTopping)". Run the reasoner. Is your VegetarianPizzaAlternative1 satisfiable What are the inferred super-classes? Which of the following are examples of inferred subclasses?
- Task 6: Defining VegetarianPizzaAlternative2. Select Thing in the class hierarchy. Create a new class VegetarianPizzaAlternative2 (use exp2). Define it as "Pizza and (hasTopping only (CheeseTopping or FruitTopping or HerbSpiceTopping or NutTopping or SauceTopping or VegetableTopping))". Run the reasoner. Which of the following classes are equivalent to VegetarianPizzaAlternative2?
- Task 7: Changing VegetarianPizzaAlternative2. Open the definition of VegetarianPizzaAlternative2 (use exp2). Remove the "SauceTopping" from the list of disjuncts (i.e. delete "or SauceTopping" from the expression), so that VegetarianPizzaAlternative2 is defined as Pizza and (hasTopping only (CheeseTopping or FruitTopping or HerbSpiceTopping or NutTopping or VegetableTopping)). Run the reasoner. Which of the following classes are equivalent to VegetarianPizzaAlternative2?
- Task 8: Meat and Veggies. Use exp2 and navigate to MeatTopping. Delete the axiom stating that MeatTopping is disjoint with VegetableTopping. Run the reasoner. What changes to the class hierarchy did you observe?
- Task 9: Adding individual Pizzas. Take a look at the individuals tab (exp5). Perhaps you realised that the individual p2 is inferred to be equivalent to the individual p4. Both pizza individuals are inferred to be MeatyPizza because of that. In order to fix this, go to exp5, click on p2, and change "hasTopping some top_tomato4" to "hasTopping some top_tomato2". Run the reasoner. Is p2 still inferred to be a MeatyPizza?
- Task 10: Italian Ingredients. We take the stance that all ingredients on true italian pizzas are also italian. In order to model this, we open the "Object Properties" tab, click on "hasCountryOfOrigin" and add in a role restriction (SuperProperty Of (Chain): "isToppingOf o hasCountryOfOrigin". Run the reasoner. Which individual toppings are now inferred to have an italian origin?

Appendix F. Tasks main study

For a more detailed breakdown of the questions, please refer to the supplementary materials.

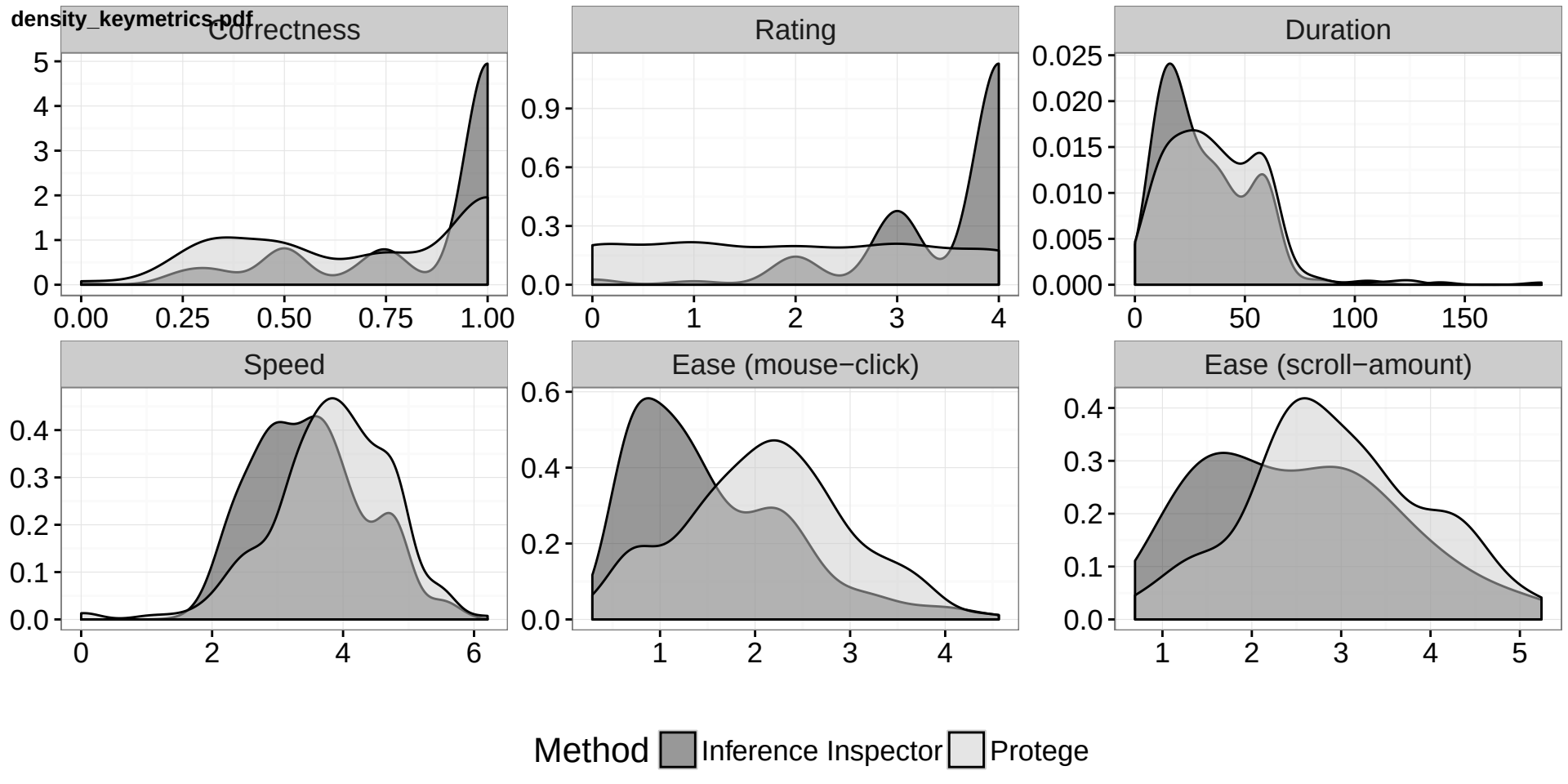
In the first scenario, you will play the role of an ontology author and interact with the well known pizza ontology. Only run the reasoner when told to do so!

- Task: Let's first find out what the ontology is about. Please run the reasoner.
  - Please acquaint yourself with the ontology for a minute, until you understand what it is about. What does it model?
  - Which of the following classes are unsatisfiable?
  - Which of the following axioms are part of the justification for the unsatisfiability of CheeseyVegetableTopping?
- Task: IceCream seems to be broken as well..
  - Which of the following axioms are part of the justification for the unsatisfiability of IceCream?
- Task: You have tried to fix the IceCream class by removing IceCream SubClassOf hasTopping some CajunSpiceTopping. Run the reasoner.
  - Did the change fix the IceCream class? (Make it satisfiable)
  - How well did the view help you in understanding whether you fixed it?
- Task: You have tried to fix the CheeseyVegetableTopping class by removing CheeseyVegetableTopping SubClassOf: CheeseTopping. Run the reasoner.
  - Did the change fix the CheeseyVegetableTopping class? (Make it satisfiable) How well did the view help you in understanding whether you fixed it? (It doesn't matter whether you actually did)
- Task: YourFavouritePizza has been falsely defined to (hasTopping some CheeseTopping) and (hasTopping some OliveTopping). You like cheese, but prefer meat over olives. Please run the reasoner.
  - Which of the following are new subclasses of YourFavouritePizza?
  - Which of the following are not subclasses of YourFavouritePizza?

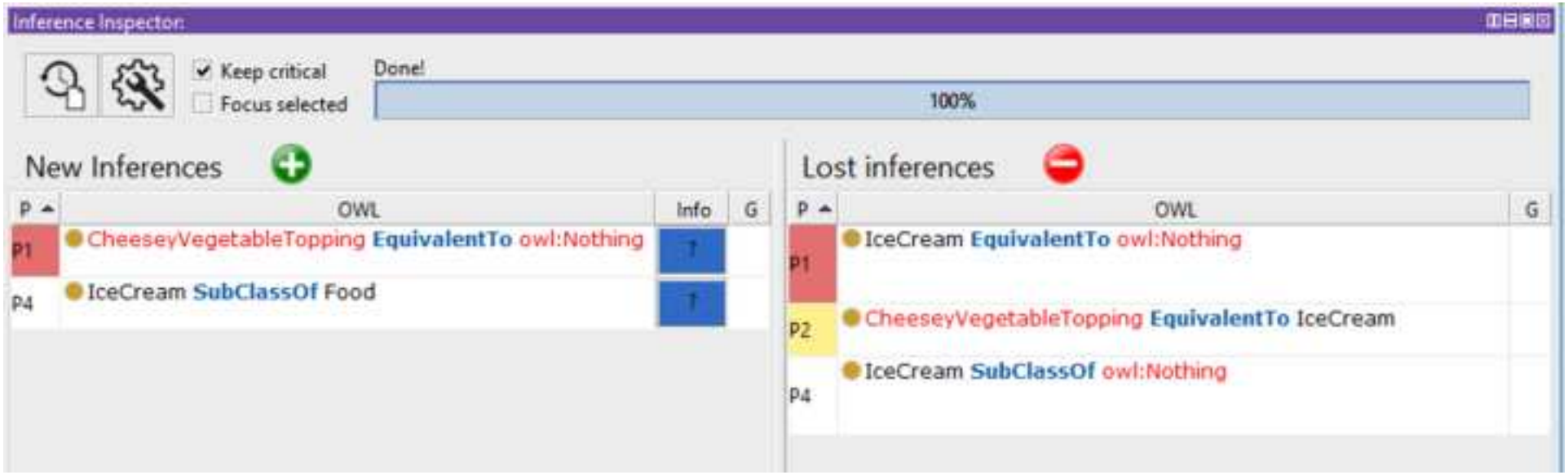
  - Which of the following are new superclasses of YourFavouritePizza? (Compared to before you changed the Definition)
- Task: Unfortunately you remembered that you decided to go vegetarian, so you define YourFavouritePizza to be a Pizza that (hasTopping some CheeseTopping) and (hasTopping some OnionTopping). Please run the reasoner.
  - Which of the following are subclasses of YourFavouritePizza?
  - Which of the following are not subclasses of YourFavouritePizza?
  - Which of the following are new superclasses of YourFavouritePizza? (Compared to before you changed the Definition)
- Task: You realise that isIngredientOf has the Domain of Food, which is perhaps underconstrained. You decide to fix this by changing the Domain to PizzaIngredient. Please run the reasoner.
  - Did the class hierarchy change?
  - Does PizzaIngredient have any new subclasses?
- Task: You feel like you should be more serious about your vegetarianism and make YourFavouritePizza disjoint with MeatyPizza. Please run the reasoner.
  - Did the class hierarchy change?
  - Are there any new unsatisfiable classes?
- Task: You decided to hit the undo button see whether everything is okay again. Please run the reasoner.
  - Did the class hierarchy change?
  - Which of the following, previously unsatisfiable classes are fixed now?
- Task: You decided to opt for a Greek interpretation of Vegetarianism and change the definition of VegetarianTopping from PizzaTopping and (not (MeatTopping)) and (not (FishTopping)) to PizzaTopping and (not (MeatTopping)). Please run the reasoner.
  - Did the class hierarchy change?
  - Which of the following classes were are equivalent to VegetarianPizza before, but are not anymore?

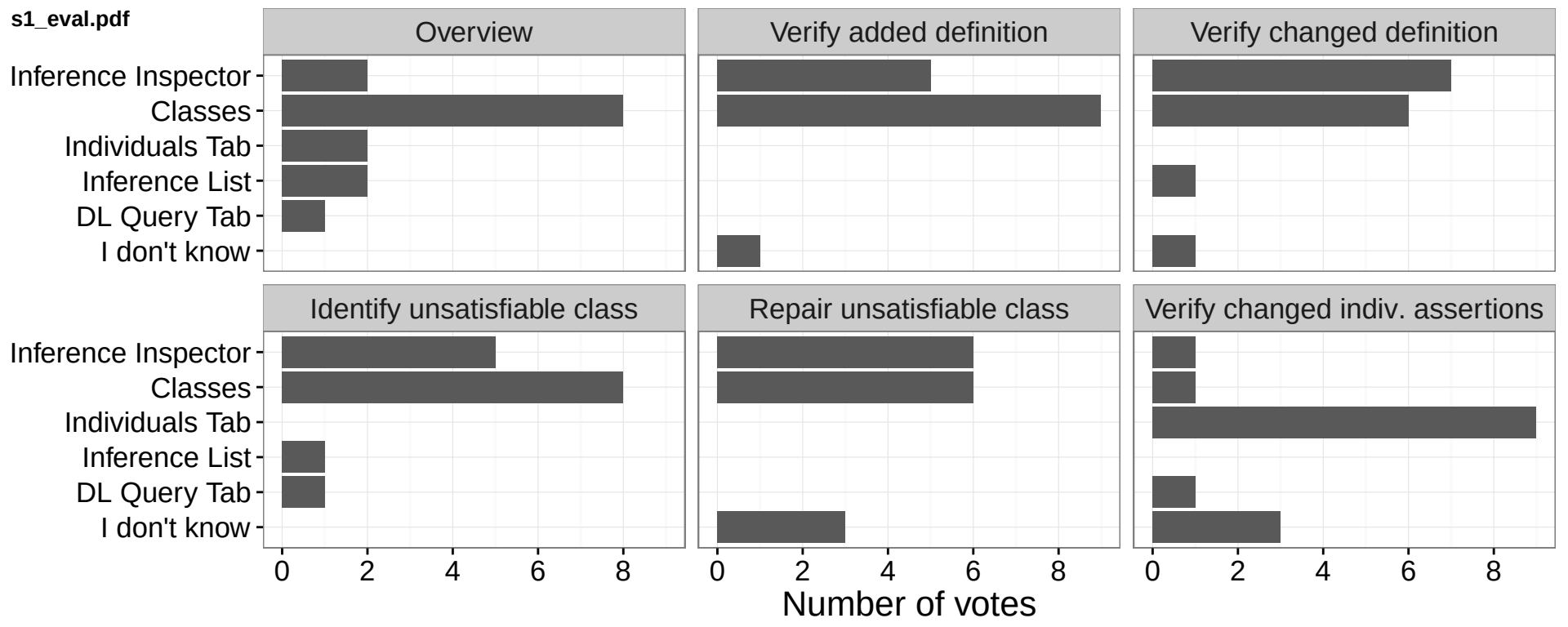
In the second scenario, you will play the role of a historian specialised in Robert Stevens Genealogy. Only run the reasoner when told to do so!

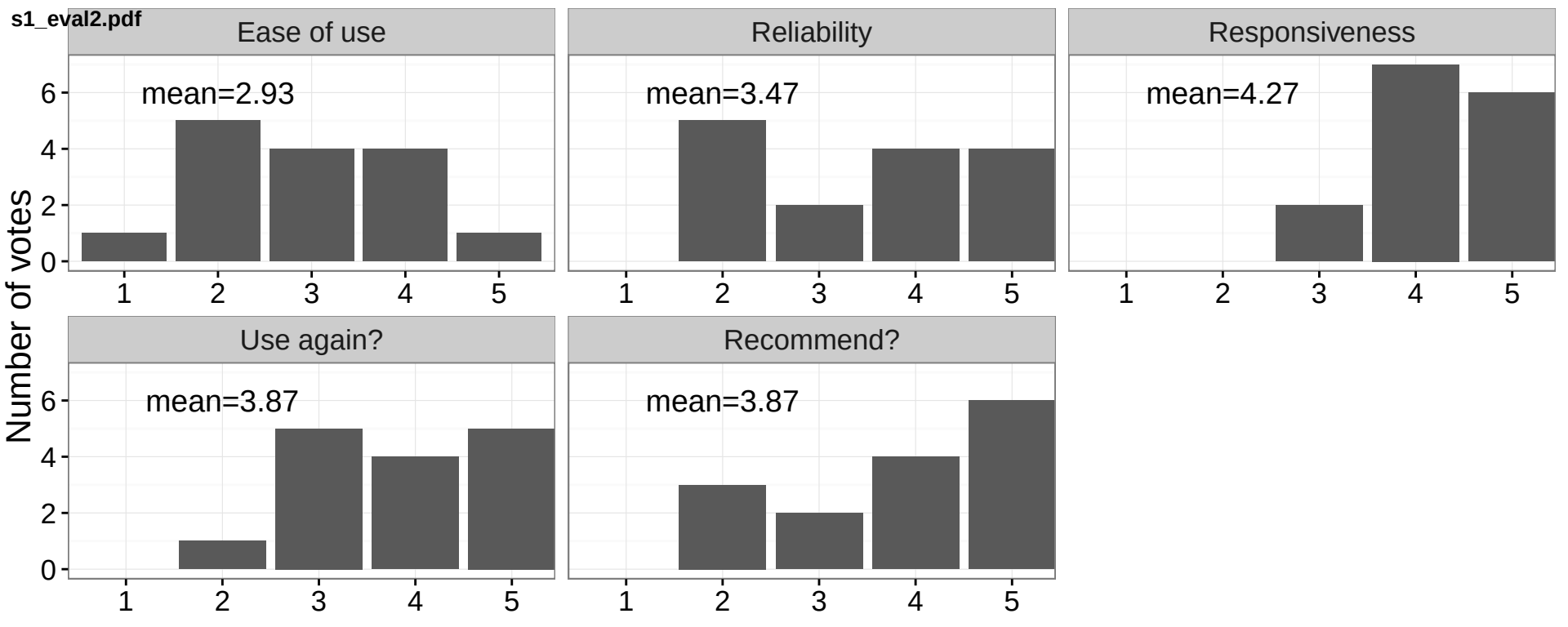
- Task: Let's first find out what the ontology is about. Please run the reasoner.
  - Are there any unsatisfiable classes?
  - Please acquaint yourself with the ontology for a minute, until you understand what it is about. What does it model?
- Task: In order to infer the type Man for some of the individuals in our knowledge base, we decide to assert that hasFather has a Range of Man. Please run the reasoner.
  - Did any individuals change their type as a result of your modelling?
  - Which of the following are true?
- Task: Perhaps you realised by now that the FemaleAncestor class is wrongly defined as Man and (isAncestorOf some Person). You have replaced Man by Woman. Please run the reasoner. Did any individuals change their type as a result of your modelling?
  - Which of the following are true?
- Task: You want to infer Siblings. You have decided to the the property-chain hasSibling SubPropertyChain: hasParent o isParentOf to infer sibling relationships. Please run the reasoner.
  - Are any individuals inferred to be siblings as a result of your modelling?
  - Which of the following are true?
- Task: You want to infer cousins. You have decided to the the property-chain hasCousin SubPropertyChain: hasParent o hasSibling o isParentOf to infer cousin relationships. Please run the reasoner.
  - Are any individuals inferred to be cousins as a result of your modelling?
  - Which of the following are true?

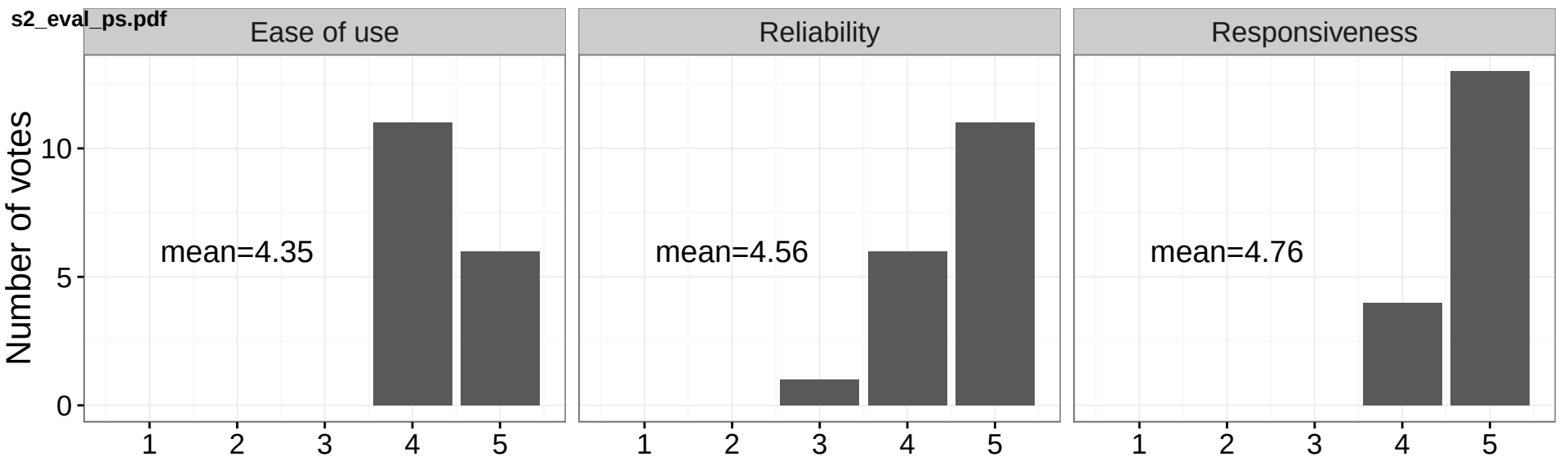

density_keymetrics.pdf
Correctness
Rating
Duration
Speed
Ease (mouse−click)
Ease (scroll−amount)
Method
Inference Inspector
Protege

inference_inspector.png




Overview
Verify added definition
Verify changed definition
Identify unsatisfiable class
Repair unsatisfiable class
Verify changed indiv. assertions
Inference Inspector
Classes
Individuals Tab
Inference List
DL Query Tab
I don't know
0
2
4
6
8
Number of votes


Ease of use
mean=2.93
Reliability
mean=3.47
Responsiveness
mean=4.27
Use again?
mean=3.87
Recommend?
mean=3.87
Number of votes
0
2
4
6
1
2
3
4
5


Ease of use
Reliability
Responsiveness
Number of votes
0
5
10
mean=4.35
mean=4.56
mean=4.76
1
2
3
4
5