Whelk: An OWL EL+RL Reasoner Enabling New Use Cases

James P. Balhoff[1]
Renaissance Computing Institute, University of North Carolina, Chapel Hill, NC, USA

Christopher J. Mungall
Lawrence Berkeley National Laboratory, Berkeley, CA, USA

Abstract

Many tasks in the biosciences rely on reasoning with large OWL terminologies (Tboxes), often combined with even larger databases. In particular, a common task is retrieval queries that utilize relational expressions; for example, "find all genes expressed in the brain or any part of the brain". Automated reasoning on these ontologies typically relies on scalable reasoners targeting the EL subset of OWL, such as ELK. While the introduction of ELK has been transformative in the incorporation of reasoning into bio-ontology quality control and production pipelines, we have encountered limitations when applying it to use cases involving high throughput query answering or reasoning about datasets describing instances (Aboxes).

Whelk is a fast OWL reasoner for combined EL+RL reasoning. As such, it is particularly useful for many biological ontology tasks, particularly those characterized by large Tboxes using the EL subset of OWL, combined with Aboxes targeting the RL subset of OWL. Whelk is implemented in Scala and utilizes immutable functional data structures, which provides advantages when performing incremental or dynamic reasoning tasks. Whelk supports querying complex class expressions at a substantially greater rate than ELK, and can answer queries or perform incremental reasoning tasks in parallel, enabling novel applications of OWL reasoning.

2012 ACM Subject Classification Information systems → Web Ontology Language (OWL); Software and its engineering → Software libraries and repositories; Applied computing → Life and medical sciences

Keywords and phrases Web Ontology Language, OWL, Semantic Web, ontology, reasoner

Digital Object Identifier 10.4230/TGDK.2.2.7

Category Resource Paper

Supplementary Material *Software (Source code)*: `https://github.com/INCATools/whelk` [9]
archived at `swh:1:dir:f8919707e053212b2c74bc63988a06ffe03fb796`
Software (Evaluation scripts and input ontologies): `https://doi.org/10.5281/zenodo.13891879` [7]
InteractiveResource (Whelk on the Web): `https://balhoff.github.io/whelk-web/`

Acknowledgements We would like to thank N.L. Harris for helpful comments on a draft of the paper.

Received 2024-07-01 **Accepted** 2024-11-10 **Published** 2024-12-18

Editors Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

Special Issue Resources for Graph Data and Knowledge

1 Introduction

OWL (Web Ontology Language) is heavily used in the biosciences as a framework for constructing widely used ontologies, for example the Gene Ontology [17], Uberon [30], Human Phenotype Ontology [16], SNOMED-CT [22], and the NCI Thesaurus [35]. These ontologies are characterized by a number of features: (1) their large size relative to other ontologies; (2) the use of existential

[1] Corresponding author

restrictions to encode graph-oriented information such as partonomies and developmental lineages (we call this the Relation Graph); and (3) the fact that the majority of axioms are encoded using the EL subset of OWL [42]. As such, they are amenable to automated reasoning using engines that are tuned for this profile, the most notable of which is the ELK Reasoner, introduced in 2010 [24]. In fact the introduction of ELK has been instrumental in making reasoning scalable for bio-ontologies [28]. ELK is the default reasoner in the ROBOT tool [23], and is deployed as a part of the production pipelines for dozens of ontologies [27].

However, two limitations of ELK constrain its applications: (1) It has limited support for instance graphs (Aboxes) in that, while it infers class membership, it does not materialize inferred object property assertions. Nor does it implement SWRL [21], a language for extending OWL with custom reasoning rules. Additionally, the expressivity of the OWL EL profile excludes key types of object property axioms, such as inverse and functional properties. Nonetheless, most available OWL reasoners with support for rich Abox reasoning do not scale to the size and complexity of terminologies easily handled by ELK. (2) Classification and query answering are blocking operations on a mutable state. That is, although ELK supports incremental reasoning, after new axioms are added and classified, querying the previous state of the ontology requires removing those just added. Likewise, because answering complex queries typically requires some incremental classification, ELK processes a single query at a time. Reasoner queries are frequently used to explore biomedical terminologies, e.g., find all classes that are a $MuscleOrgan\ and\ (partOf\ some\ Head)$.

In order to support various use cases hindered by those limitations, we implemented the Whelk reasoner, based on the ELK algorithm described by Kazakov et al [24]. In addition to the OWL EL reasoning rules defined for ELK, Whelk supports the OWL RL profile [42] as well as reasoning with SWRL rules. Whelk's implementation is based on immutable functional data structures [32], so that each time axioms are added, a new reasoner state is created; references to the previous state remain unchanged. This allows Whelk to answer queries concurrently, and also allows concurrent incremental classification of unrelated sets of axioms – for example, various Abox ontologies which build upon the same ontology Tbox. The Tbox can be classified ahead of time and reused. We evaluated Whelk on varied benchmarks taken from real-world ontology reasoning scenarios, comparing Whelk to two releases of ELK, the state of the art open source OWL EL reasoner.

2 OWL ontologies

OWL is an ontology language for the Semantic Web [40]. An ontology is a formal representation of concepts within a domain and the logical relationships between those concepts, supporting knowledge representation with explicit semantics. The semantics of OWL are based in Description Logics [2]. The format we use for OWL examples in this paper follows the user-friendly Manchester syntax [41]. An OWL ontology models a domain using classes, properties, and individuals, and consists of axioms making statements about these entities. Individuals are specific objects within the model (e.g., Bob, $Alice$, $NewYorkCity$), while classes are used to define categories of individuals (e.g., $Person$, $City$). Individuals are said to be instances of classes. Properties denote relationships between individuals in the model (e.g., $Bob\ livesIn\ NewYorkCity$). In addition to simple named classes, complex class expressions can be constructed that define categories based on combinations of other class expressions and properties. For example, the concept "all entities that live in some City" can be constructed using an expression known as an existential restriction, which describes a relationship which all members of the class must have (in Manchester syntax, $livesIn\ some\ City$). By constructing the intersection of that expression with the class $Person$, we can create an expression representing the class of city dwellers: $Person\ and\ (livesIn\ some\ City)$. Named classes, e.g., *CityDweller*, can be linked to expressions defining them using an equivalent class axiom: *CityDweller EquivalentTo* (*Person and* (*livesIn some City*)). Classes can also be related hierarchically to one another via a subclass axiom (e.g., *City SubClassOf GeographicalRegion*), and declared to have no instances in common (e.g., *GeographicalRegion DisjointWith Person*). Likewise, OWL supports hierarchical relationships among properties, as well as property features such as transitivity and property chains, which allow us to infer new relationships between individuals that are linked by a sequence of intervening relationships. A familiar property chain example would be inference of *hasUncle* from a path: $hasParent \circ hasBrother \rightarrow hasUncle$.

An OWL reasoner can be used to perform a number of tasks with regard to an OWL ontology, by computing the implied consequences of the asserted axioms. One of the most common tasks is Tbox (terminology) *classification*, that is, computing the hierarchical relationships (subsumptions) among all classes in the ontology. Checking if any classes are inferred to be equivalent to `owl:Nothing` (the empty class) is a common quality control task for OWL-based terminologies. Another common task is Abox (assertions) *materialization*: computing all inferred relationships between individuals, as well as the inferred types of the individuals (their class membership). *Consistency checking* evaluates whether the ontology contains a logical impossibility, based on the provided axioms. Different OWL applications may focus only on a subset of reasoning tasks. Development of large bio-ontologies is often solely focused on classification. Developers of an anatomical terminology may make heavy use of class expressions and inferred subsumptions to ensure consistent modeling and automatic calculation of a complex hierarchy, but never create instances of the terms. On the other hand, other ontology use cases may be more focused on instance graphs, computing inferred relationships between individuals, possibly using a less complex schema-like terminology. An OWL reasoner may also be used to answer queries over the inferred knowledge, returning all subclasses known for a given class, or all individuals which are instances of a given class. Queries using complex class expressions are commonly referred to as DL (description logic) queries.

Even though the complete OWL DL language is decidable, in practice for many tasks DL reasoning over ontologies of non-trivial size is time and compute-intensive, and often infeasible. For this reason OWL provides a number of profiles (language subsets), each of which limits the language in specific ways in order to allow more efficient reasoning. The profiles are designed to provide adequate expressivity for particular use cases. For example, the OWL EL profile provides logical features commonly used in the development of large complex terminologies, such as bio-ontologies, where the focus is ontology classification based on existential restrictions and intersections, and quality control using disjoint classes axioms. The OWL RL profile is more targeted to inference of relationships among large numbers of individuals, providing additional property features such as inverse properties and functional properties, while at the same time having fewer features for inferring class hierarchies as compared to EL.

3 Features and implementation

Whelk is implemented in Scala, a programming language for the Java Virtual Machine (JVM), which is fully interoperable with Java and has strong support for functional programming, plus language constructs that encourage immutable data structures [44]. Whelk provides an implementation of the `OWLReasoner` interface defined by the Java OWL API [20], making it readily usable within popular software for working with OWL such as Protégé [31] and ROBOT [23]. Whelk can also be used within pure Scala programs without reliance on the OWL API. Via Scala.js [45], it can be used as part of browser-based JavaScript applications, and can also be compiled to native code using Scala Native [43].

Whelk supports all axiom types within the OWL EL and RL profiles [42], with the limitation that reasoning about data property values (concrete values such as strings or numbers) is not supported. Additionally, `HasKey` axioms are not supported. Whelk also extends OWL EL and RL reasoning with SWRL rules (again with the exception of data property values). SWRL rules allow matching arbitrary patterns of class assertions and object property assertions to generate new inferred class and object property assertions about individuals.

3.1 Parallel extension of reasoning state

As described above, Whelk is built on immutable data structures which return a new instance when modified, rather than allowing mutation. Implementations based on shared structure allow reasonable performance and efficient use of memory [32]. A Whelk reasoner instance is initialized with a starting set of axioms. All reasoning rules are applied, and a reasoning state object is returned containing the classification derived up to that point. This reasoning state can be extended with additional asserted axioms. Reasoning continues until classification is complete, and a new reasoning state is returned. Any references to the earlier reasoning state are still valid, and can be queried without reflecting conclusions derived from extension with the second set of axioms. Thus any number of independent extensions to the reasoning state can be created. This is much like programming with a singly linked list in a language like Lisp or Haskell; prepending a new item to a list of size 2 results in a new list of size 3, but doesn't affect references to the first list, which still consists of 2 elements.

When answering a DL query, the reasoning state is extended with an equivalent class axiom representing the query expression. Once additional reasoning is completed and the query is answered, the new reasoning state can simply be discarded in order to roll back to the initial state. This approach provides Whelk with very fast DL query performance. Since the reasoning state is immutable, any number of queries can be processed simultaneously without interference. Currently, while any kind of axiom can be provided in the starting set, only class (Tbox) and individual (Abox) axioms are supported when extending reasoning states. Adding new property (Rbox) axioms or SWRL rules after the initial classification is complete requires a complete reclassification (a limitation shared with ELK). The approach described for processing DL queries works equally well for other use cases, such as extending a reasoning state representing a large terminology with various sets of Abox axioms describing individuals instantiating that terminology, or applying sets of additional Tbox axioms representing alternative conceptions of a domain.

3.2 Supported reasoning tasks

Like ELK, Whelk supports standard OWL reasoning tasks such as classification, coherency and consistency checking, and queries for subclasses, superclasses, or instances of arbitrary class expressions. In addition, Whelk is also able to materialize all inferred relationships between individuals (object property assertions), a feature not directly supported by ELK.

3.3 Reasoning implementation

3.3.1 OWL EL

Whelk's EL reasoning is based on the inference rules detailed in figure 3 of Kazakov et al.[24]. Although the concrete implementation looks quite different from the source code of ELK due to the choice of language and use of immutable data structures, Whelk's implementation closely follows Algorithm 2 of Kazakov et al., taking an expression from the queue and applying each rule in turn, adding any generated expressions to the queue. Whelk's Scala code is compact and attempts to transform the ELK rules to programming code as directly as possible. As one example, ELK's rule $R^-_\sqcap$ handles the case that a concept that is a subclass of an intersection expression should be inferred to be a subclass of each of the concepts in the intersection:

$$\frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}$$

The Scala version of this rule is a function that accepts a concept inclusion (subclass inference) from the queue, along with the current reasoner state (which holds various indices providing fast lookup into the ontology and computed inferences) and the queue collecting produced expressions to which rules will be applied. If the superclass in the concept inclusion is an intersection (called a `Conjunction` in the data model), then two new concept inclusions are added to the queue:

```
def ruleMinConj(
  ci: ConceptInclusion,
  reasoner: ReasonerState,
  todo: Stack[QueueExpression]): ReasonerState =
  ci match {
    case ConceptInclusion(sub, Conjunction(left, right)) =>
      todo.push(ConceptInclusion(sub, left))
      todo.push(ConceptInclusion(sub, right))
      reasoner
    case _                                               => reasoner
  }
```

ELK 0.6.0 improved its coverage of OWL EL by adding support for property range axioms. The latest release of Whelk also supports the use of property ranges. ELK currently lacks complete support for "self restrictions", that is, class expressions stating that all instances of that class have a specific property relation to themselves. These expressions are fully supported by Whelk, which enables the use of a useful pattern known as rolification [25], in which a property acts as a sort of marker for a class. As one example, the OBO Relation Ontology (RO) includes the following axioms defining a rolification property for the class *KinaseActivity* and using it in a property chain:

$$KinaseActivity\ SubClassOf\ (isKinaseActivity\ some\ Self)$$

$$capableOf\ \circ\ isKinaseActivity\ \circ\ hasDirectInput\ \rightarrow\ phosphorylates$$

This in effect defines a property chain $capableOf\ \circ\ hasDirectInput$ that is only matched when the intermediate node has the type *KinaseActivity*.

3.3.2 OWL RL

As stated above, Whelk extends the ELK design with support for the OWL RL profile. Handling of OWL RL axiom types that are not included within the OWL EL profile is accomplished via three approaches. First, certain OWL RL axioms, while not explicitly included in OWL EL, can be transformed to equivalent EL constructs. For example, OWL EL does not include union class expressions (e.g., *Animal or Plant*). These expressions are allowable within OWL RL, but only when used on the subclass side of subclass axioms. By asserting a subclass for such expressions for each of the union operands when loading the ontology, we can obtain the inferences supported by OWL RL using the ELK reasoning rules. Thus, if an expression such as C *or* D appears in the ontology, we need only to inject these axioms:

$$C\ SubClassOf\ (C\ or\ D)$$

$D\ SubClassOf\ (C\ or\ D)$

For OWL RL complement expressions, e.g., *not C*, we inject:

$(C\ and\ (not\ C))\ SubClassOf\ owl{:}Nothing$

And for OWL RL cardinality restrictions of cardinality 0, e.g., *p max* 0 *C*, we inject:

$(p\ max\ 0\ C)\ and\ (p\ some\ C)\ SubClassOf\ owl{:}Nothing$

The first two transformations are also supported by ELK [14].

Secondly, support for the remaining OWL RL class expression constructs is provided by additional rules implemented similarly to the standard ELK rules. These include "all values from" restrictions and cardinality restrictions of cardinality 1. For example, the rule for "all values from" can be written in the style used in Kazakov et al., where `i` and `j` are individuals:

$$\frac{i \sqsubseteq \forall R.C \quad i \xrightarrow{R} j}{j \sqsubseteq C}$$

In this rule, $i \xrightarrow{R} j$ is a *link*, a type of conclusion representing existential restrictions used in the ELK reasoning rules; a link between two individuals is equivalent to an object property assertion. As part of Abox materialization, Whelk generates all inferred links between individuals, so that there is no need for this rule to consider the property hierarchy. This rule is implemented by two Scala functions similar to the above example: one to handle newly generated concept inclusions, and one to handle newly generated links.

Lastly, other OWL RL axiom types are transformed to equivalent SWRL rules, and handled by Whelk's SWRL rule engine. For example, the inverse properties axiom `p inverseOf r` is transformed to these SWRL rules, where leading question marks indicate variables:

$p(?x, ?y) \rightarrow r(?y, ?x)$

$r(?x, ?y) \rightarrow p(?y, ?x)$

3.3.3 SWRL rule engine

The Whelk SWRL rule engine is implemented as an extension of the EL reasoner. It supports inference based on user-defined rules which can match patterns of individual types and relationships (as above, reasoning with datatype property values is not currently supported). An example of such a rule included in the OBO Relation Ontology is one that infers "phosphorylates" relationships between gene product instances:

$$directlyRegulates(?a1, ?a2)\ \wedge\ KinaseActivity(?a1)\ \wedge\ enabledBy(?a1, ?g1)$$
$$\wedge\ enabledBy(?a2, ?g2) \rightarrow phosphorylates(?g1, ?g2)$$

SWRL rules perform instance-level reasoning; that is, they match and produce class assertions and property assertions for individuals. As stated above, Whelk generates SWRL rules for certain OWL RL axioms; these ensure that all inferred relations between individuals are materialized. Like the EL reasoner, the SWRL rule engine works on a queue of produced expressions. When an expression is taken from the EL reasoner queue, if it is a concept inclusion involving an individual, or a link where the subject and object are individuals, it is also added to the SWRL engine queue. The SWRL rule engine is an implementation of the widely used Rete pattern-matching algorithm first developed by Forgy [15] and described in detail by Doorenbos [13]. Its design is an adaptation of our earlier work on an RDF rule engine [4]. When the rule engine is constructed, it builds a tree of join nodes, where each node represents a pattern occurring in the body of a rule, linked in such a way that a path from the root to a leaf represents a complete rule body. As concept inclusions and links are taken from the queue, they are sent to any join nodes which use the same class or property predicate. Join nodes in turn check their predecessor join nodes (if any) for compatible partial solutions, and if found, activate successor join nodes. The final node in a tree branch is a production node representing a rule head, which may generate a new concept inclusion or link (representing class assertions and object property assertions), which is added to the EL reasoner queue.

This integration does result in some redundant derivation of inferences. For example, while the ELK reasoning algorithm handles transitive properties and property chains, it attempts to derive only the links required for complete classification of named classes in the ontology [24]. In order to compute a complete set of OWL RL inferences and to materialize all inferred object property assertions, we additionally create SWRL rules for transitive properties and property chains for use in the rule engine. While there may exist a more efficient approach, the integration nonetheless provides the capability for computing Abox inferences while working with terminologies requiring an OWL EL reasoner for scalability, a feature not available from most EL reasoners.

4 Evaluation

4.1 Testing

The Whelk codebase includes a suite of unit tests to verify that it derives identical subsumptions to ELK for OWL EL axioms. Inferences for OWL RL are compared to the output of the OWL reasoner HermiT [18] (which supports all OWL features but is not scalable for large ontologies) using test cases targeting the reasoning rules outlined in the OWL RL profile spec [42]. The test suite can be easily extended by adding new test ontologies to the repository.

4.2 Performance

We compared Whelk 1.2.1 to two versions of ELK: the long-established ELK 0.4.3, and the very recently released ELK 0.6.0, which adds support for object property range axioms. In the performance evaluations we make use of three ontologies (Table 1):

- *UNIV-BENCH-OWL2EL* – a benchmark Tbox covering the OWL EL profile [34].
- *uberon-go-cl-ro* – a merged set of ontologies from the OBO Foundry containing mutually referential axioms: Uberon anatomy ontology, Gene Ontology (GO), Cell Ontology (CL), Relation Ontology (RO) [30, 17, 12, 36]. Because the published versions of OBO ontologies typically include the precomputed classification, our test ontology was derived from the source version of each component ontology.
- *nci-thesaurus* – the NCI Thesaurus reference terminology from the National Cancer Institute [35].

Each performance evaluation was implemented as a custom Scala script. All testing scripts and input ontologies are available from the whelk-paper GitHub repository archived at `https://doi.org/10.5281/zenodo.13891879`. All performance tests were executed on an Apple MacBook Pro with an M2 Max chip with 12 cores, and 64 GB RAM. We set the maximum heap size for the Java Virtual Machine to 16 GB.

Table 1 Characteristics of test ontologies.

Ontology	Classes	Object properties	Logical axioms
UNIV-BENCH-OWL2EL	131	82	398
uberon-go-cl-ro	70,057	656	108,610
nci-thesaurus	188,034	97	258,241

4.2.1 Computed subsumptions

Because the three reasoners differ slightly in support for axiom types, we filtered out axioms making use of `ObjectUnionOf` and `ObjectHasSelf` expressions as well as `ObjectPropertyRange` axioms. We programmatically verified that all three reasoners derived the same class subsumptions for each test ontology.

4.2.2 Ontology classification speed

We measured the time required for each reasoner to classify each of the input ontologies and answer a query to check the coherency of the ontology ("list any classes equivalent to `owl:Nothing`"). Within a single process for each reasoner and ontology combination, we computed the classification of each ontology 12 times. We discarded the first two runs (to allow warmup of the JVM) and computed the average of the remaining 10 runs. ELK's classification algorithm is multi-threaded, while Whelk's is single-threaded only; we allowed ELK to use all available CPUs. The UNIV-BENCH-OWL2EL ontology is so tiny that each reasoner takes only a fraction of a second, and so we exclude it from further performance tests. For classification of nci-thesaurus, ELK 0.4.3 is ~10 times faster than Whelk, completing the task in 2.4 seconds vs. 24.2 seconds for Whelk (Figure 1). On the other hand, ELK 0.6.0 is significantly slower than its previous release, comparable to Whelk with a time of 23.2 seconds. For uberon-go-cl-ro, ELK 0.4.3 outperforms Whelk by a larger margin (~20 times faster), at 2.3 seconds vs. 46.8 seconds. ELK 0.6.0 is the slowest for uberon-go-cl-ro, averaging 99.7 seconds. The decrease in performance between ELK 0.4.3 and 0.6.0 is unexpected, and we plan to investigate this issue with the developers.

4.2.3 DL query speed

In order to measure how quickly each reasoner can answer successive DL queries, for each ontology (uberon-go-cl-ro and nci-thesaurus) we extracted all complex class expressions used, at all levels of nesting. This procedure provides us with class expressions likely to represent relevant queries, rather than generating random combinations of classes and properties. After classifying the ontology, we queried the reasoner for subclasses of each class expression, using the 'getSubclasses' method of the `OWLReasoner` interface. As assurance that each reasoner returned the same subclasses, we collected the number of subclasses returned for each class expression and reported the sum. The query script submitted queries to the reasoner at each of three levels of parallelism: 1 (query all expressions sequentially), 4, or 8 workers. We measured the time required to answer all queries for all combinations of ontology, reasoner, and parallelism, averaging three runs following a warmup run.

For sequential queries, ELK 0.4.3 and 0.6.0 perform similarly, executing 46,685 queries against nci-thesaurus at 51 and 47 queries per second, respectively (Figure 2). In comparison, Whelk is able to execute 2306 queries per second. As expected, at higher levels of concurrent queries, ELK's performance remains the same. Whelk's query answering speed scales with the number of workers: 8159 queries per second using 4 workers; 13,679 queries per second using 8 workers.

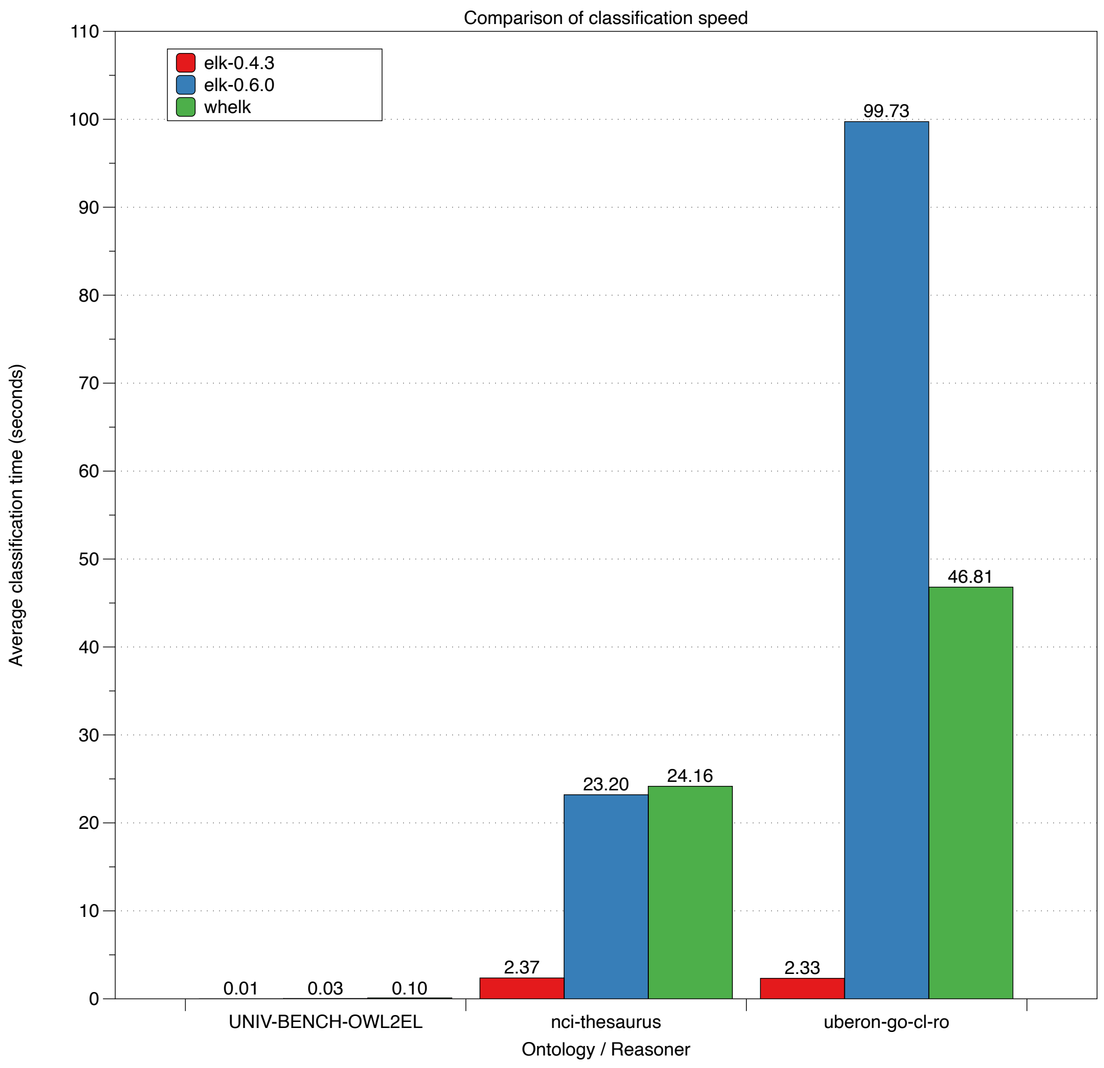

Figure 1 Time required for each reasoner to classify and check coherency of three input ontologies (lower is better). ELK 0.4.3 is 10-20 times faster than Whelk for the two large ontologies that we tested (nci-thesaurus and uberon-go-cl-ro).

Results for executing 66,169 queries against uberon-go-cl-ro follow the same pattern, although in this case ELK 0.6.0 outperforms 0.4.3 (94 queries per second vs. 24, for sequential queries). Again Whelk is much faster at 1397 queries per second (sequentially) and 8735 queries per second (8 workers).

4.2.4 Abox consistency checking speed

We next tested how quickly each reasoner could perform incremental reasoning when extending an ontology with multiple independent sets of Abox axioms, after having previously classified the Tbox. We used the uberon-go-cl-ro ontology as a Tbox for a collection of 4590 Gene Ontology Causal Activity models (GO-CAMs) [37]. This ontology contains axioms defining most of the biological concepts and relations used in the GO-CAMs. The GO-CAMs are small Abox ontologies of class assertions and object property assertions, containing an average of 43 logical axioms, with

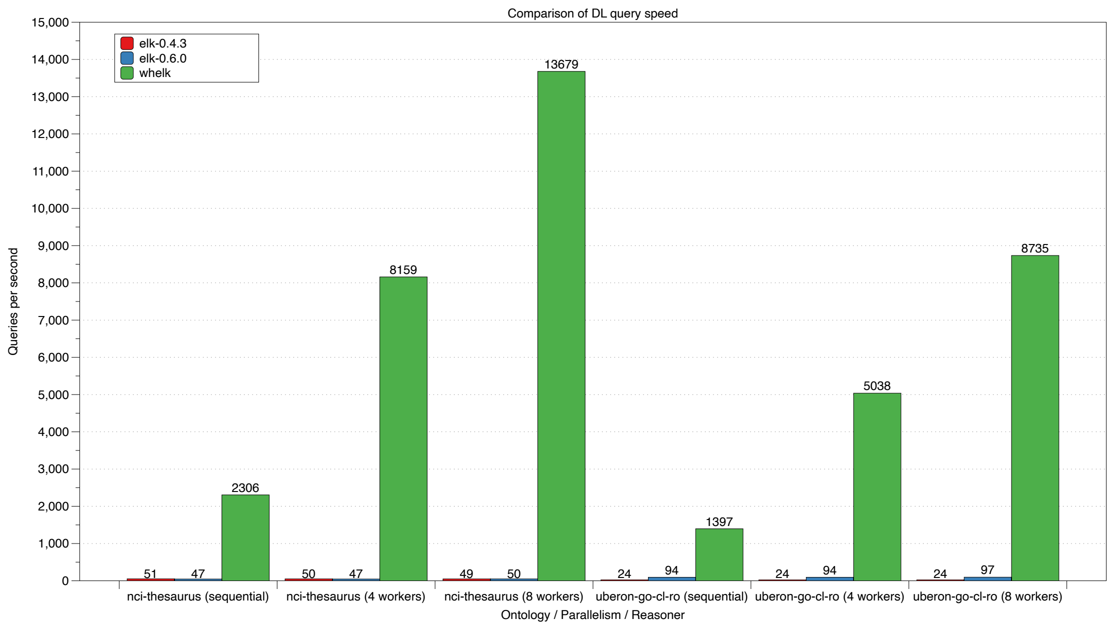

Figure 2 Query answering rate at different levels of parallelism of DL query submission (higher is better). ELK processes a single query at a time, while Whelk responds to parallel requests. Whelk's sequential DL query speed is ~15-50 times greater than ELK for these two ontologies, and up to 279 times faster when handling 8 queries at a time.

the smallest containing 4 and the largest containing 2590. Similarly to the DL query test, we used each reasoner to first classify the ontology, then added each Abox either sequentially or in parallel using 4 or 8 workers. We measured the time to classify and then query the consistency of all 4590 Aboxes, averaging three runs following a warmup run. For the two ELK reasoners, for each Abox we added its axioms to the base ontology using the OWL API and used the `OWLReasoner` "flush" method to trigger classification. After querying the consistency of the result, we removed the Abox axioms from the ontology. Because Whelk is designed particularly for this scenario, we used its Scala API directly rather than going through OWL API, which does not allow adding axioms in parallel.

Both ELK reasoners found 36 inconsistent Aboxes. Whelk detected 56 inconsistent Aboxes; this difference is expected since Whelk supports OWL RL constructs that ELK does not. In the sequential scenario, ELK 0.4.3's greater classification speed allowed it to outperform the other reasoners, checking 121 Aboxes per second compared to 84 for Whelk and 28 for ELK 0.6.0 (Figure 3). As in the DL query test, performing the tasks in parallel did not provide appreciable benefit for the ELK reasoners, but allowed much higher throughput using Whelk, checking 286 Aboxes per second with 4 workers, and 498 Aboxes per second with 8 workers.

5 Applications

5.1 Relation graph materialization

A substantial number of ontology use cases in the biosciences translate to what we call "relation graph" questions, such as "what are the parts of the nucleus" or "where is this gene localized". These can be translated to OWL subclass queries involving existential restrictions, e.g. $?c$ $SubClassOf$ R $some$ D or C $SubClassOf$ R $some$ $?d$. The latter is particularly challenging,

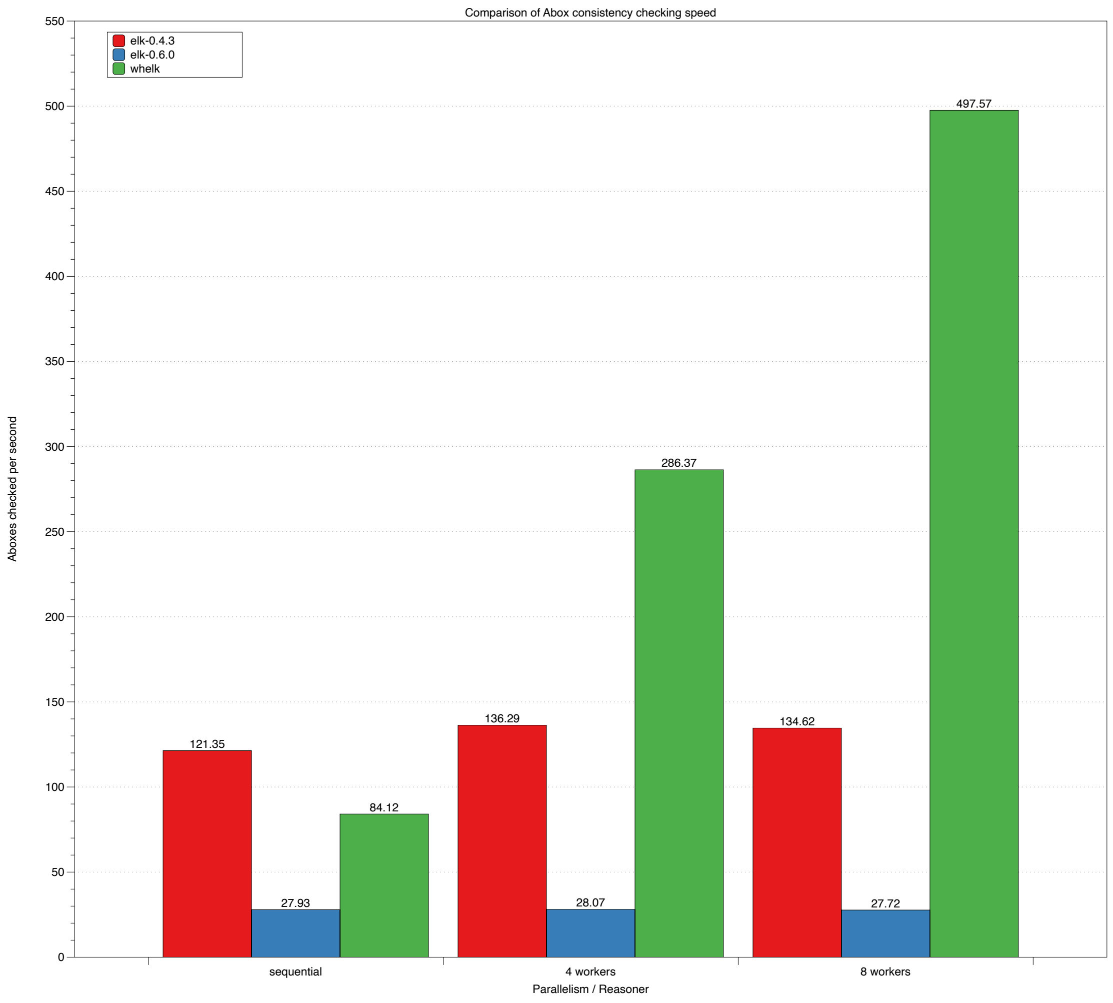

Figure 3 Abox consistency checking rate at different levels of parallelism of axiom addition (higher is better). ELK processes a single incremental reasoning task at a time, while Whelk can extend its reasoning state in parallel.

as the way to answer this with standard OWL query interfaces is to test subsumption for different values of $?d$, rather than executing a single query. A relation graph in the sense we describe is illustrated in Figure 4, which shows a small subset of the Gene Ontology TBox depicting subclass axioms and subclass existential axioms as edges.

We previously implemented a system for computing such entailed existential relation edges using the OWL API and the ELK reasoner, but we found the performance did not scale to the level we needed. Constructing and classifying named versions of all combinations of properties and classes quickly generates a prohibitively large ontology. Performing successive DL queries can be done with a manageable, constant memory size, at the cost of a possibly long runtime. Using Whelk, we created a tool called "relation-graph", which efficiently materializes every inferred relationship $C\ SubClassOf\ R\ some\ D$ for all properties and classes from an input ontology. Relation-graph relies on Whelk's much faster query answering performance, and also performs queries in parallel. Further, it makes use of the class and property hierarchies to avoid unnecessary queries. Entailed $C\ SubClassOf\ R\ some\ D$ relationships are output as simple RDF triples $C\ R\ D$, which lend themselves to straightforward SPARQL queries that are logically complete after the relation-graph materialization.

Relation-graph is a crucial component of the Ubergraph construction pipeline, which generates an RDF knowledge graph combining many OBO library ontologies along with the full set of materialized relation graph edges [3]. The Ubergraph relation closure has proved to be a valuable resource for conveniently harnessing the logical semantics provided by its component ontologies [10, 19], and relation-graph itself is used to support a number of other applications [33, 38]. As part of the relation-graph tool, Whelk has been of critical value in making the Ubergraph reasoning precomputation feasible. Ubergraph is based on a large merged ontology, presently consisting of more than 5 million logical axioms, with almost 4 million named classes and more than 1000 object properties. It takes Whelk approximately 40 minutes to classify the ontology, while ELK 0.4.3 can classify this ontology in 99 seconds. However, in the course of building Ubergraph, the relation-graph tool first classifies the ontology and then uses Whelk to execute more than 90 million DL queries. In our tests, on this ontology ELK completes ~2.5 queries per second; at that rate it would take more than 400 days to complete this task, while this phase of the Ubergraph build completes in less than 8 hours using relation-graph with Whelk, making the extended classification time a worthwhile trade-off.

5.2 Reasoning with Aboxes and biomedical terminologies in Protégé and ROBOT

Whelk is the only reasoner available for the OWL API we are aware of which can efficiently classify large biomedical ontologies such as Uberon, Gene Ontology (GO), and NCI Thesaurus and also materialize inferred object property assertions. It therefore fills a valuable niche for those who are using such ontologies to reason over instance models, for example within the Gene Ontology GO-CAM project [37]. The GO Consortium uses GO-CAMs to describe the activity of gene products within cellular processes, using OWL to provide a much more expressive modeling capability than traditional flat gene-to-term associations. The modeling in GO-CAMs relies on the rich property axiomatization in the OBO Relation Ontology, including inverse property axioms and SWRL rules. The core reference terminology, combining GO, Uberon, CL, RO, and several other ontologies, consists of nearly 1 million logical axioms. Being able to load this ontology into Protégé and classify a GO-CAM is invaluable for exploring modeling consequences or debugging unexpected inferences found in a particular model. Whelk can also be used to materialize Abox inferences within the command-line OWL tool ROBOT. This can done for one of the GO-CAM files described above with a command such as the following:

```
robot merge -i uberon-go-cl-ro.ofn -i gocam-5b91dbd100000506.ttl \
reason --reasoner whelk --axiom-generators "ClassAssertion PropertyAssertion" \
-o inferred.ttl
```

Unexpected inferences can also be debugged within ROBOT, using the “explain” feature it shares in common with Protégé.

5.3 Reasoner-driven web services

We have integrated Whelk as a reasoner option within Owlery [5], an application for exposing OWL reasoner functionalities via a set of web services. Like Protégé and ROBOT, Owlery is built upon the Java OWL API and thus integration of any reasoner supporting the OWLReasoner interface is trivial. Owlery supports standard OWLReasoner queries such as subclasses, superclasses, and equivalent classes of submitted class expressions, returning the results in a JSON-LD format. A distinct advantage to using Whelk within such a web services application is that, while the initial

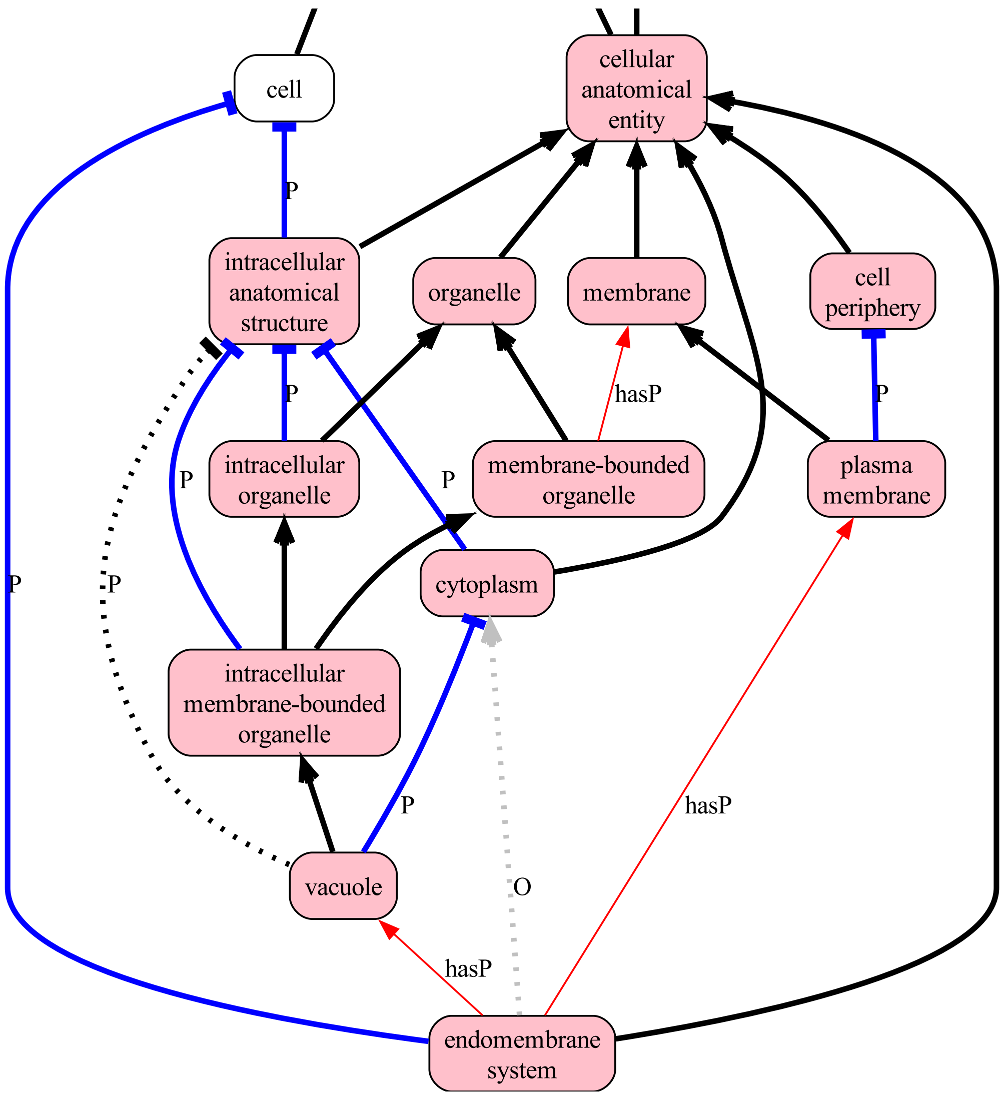

Figure 4 Subgraph of the GO Tbox focused on "endomembrane system", rendered as a relation graph. Solid lines indicate asserted axioms and dashed lines entailed. Black: *SubClassOf*; Blue (P): *C SubClassOf partOf some D*; Red (hasP)): *C SubClassOf hasPart some D*. Grey (O): *C SubClassOf overlaps some D*. The entailed dashed lines follow from the indicated axioms plus: *partOf* and *hasPart* are transitive, and there is a property chain $hasPart \circ partOf \rightarrow overlaps$.

ontology classification at startup may be slower, such a service can then run indefinitely, and subsequent queries to the reasoner are non-blocking, allowing scaling to a much higher level of concurrent traffic as demonstrated by the DL queries tests above (figure 2).

In addition to server-side web services, as noted above Whelk can be compiled to JavaScript using Scala.js. As far as we are aware, Whelk is the only OWL reasoner available for use within web browser client-side code. We provide a demonstration at `https://balhoff.github.io/whelk-web/`.

5.4 Testing hypothetical axioms

We previously implemented a system (k-BOOM [29]) for converting ontology term mappings into precise logical relationships; k-BOOM generates hypothetical axioms representing possible interpretations of a set of mappings, and attempts to find the set of interpretations which is both logically coherent and has the highest joint probability. This system was used to construct the initial version of the Mondo, a disease ontology which provides a unified logical view over several different source terminologies [39]. The original version of k-BOOM, based on ELK, required more than a day of runtime to analyze the term mapping inputs for Mondo. We have implemented a new tool, boomer [8], which is based on Whelk and can perform the same task in a matter of minutes.

6 Discussion

While ELK 0.4.3 provides much higher performance for ontology classification, Whelk's design allows it to target use cases for which ELK does not perform as well. As shown above, these use cases primarily involve concurrent extension of existing reasoner states, although even for sequential DL queries without parallelism, Whelk's design proves to result in very high performance. While our work directly reuses the rules for inference defined in the ELK publication, the success of Whelk in supporting the particular scenarios described here brings to light the value in exploring alternative reasoner implementations targeting different software ecosystems or performance use cases. Unfortunately, a recent study shows that only 25 of 73 tested OWL reasoner implementations are usable and actively maintained [1]. Many OWL reasoners have begun life as research prototypes providing a single implementation, and very few have grown into community-developed open source projects (although there are exceptions, for example Openllet [11]).

While Whelk is primarily maintained by a single developer, it is now used within a number of different applications supporting life sciences research projects making use of ontology-based knowledge graphs, which support its continued development. The Whelk codebase is fairly compact (the core EL reasoning rules comprise less than 500 lines of Scala), and we have created preliminary ports to other languages, including Rust [6], allowing it to be used with the recently developed horned-owl package [26].

Whelk's development and extensions to the ELK inference rules have been driven by a pragmatic approach. For example, the EL and RL reasoning engines derive some duplicate inferences. While we would welcome input on a more efficient integration, the current implementation returns sound results and provides useful Abox inference features as an add-on to OWL EL.

In this paper we have explored the utility of the Whelk reasoner almost entirely in contrast to its precursor, ELK. This is a testament to how universally important the ELK reasoner has been to the bio-ontology ecosystem, allowing reasoner-driven quality control pipelines to become the norm for many widely used ontologies. ELK still remains vitally important for those workflows, as well as for most interactive terminology development within Protégé. But for the additional types of use cases described here, Whelk provides distinct advantages.

References

1. Konrad Abicht. Owl reasoners still useable in 2023, 2023. doi:10.48550/arXiv.2309.06888.
2. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2 edition, 2007.
3. J Balhoff, Ugur Bayindir, Anita R Caron, N Matentzoglu, David Osumi-Sutherland, and C Mungall. Ubergraph: integrating OBO ontologies into a unified semantic graph. In *ICBO 2022: International Conference on Biomedical Ontology (ICBO)*, 2022. doi:10.5281/zenodo.7249759.

4 J Balhoff, Benjamin M Good, S Carbon, and C Mungall. Arachne: An OWL RL reasoner applied to gene ontology causal activity models (and beyond). In *17th International Semantic Web Conference (ISWC 2018)*, October 2018. doi:10.5281/zenodo.2397192.

5 James P Balhoff. owlery: Owlery is a set of REST web services which allow querying of an OWL reasoner containing a configured set of ontologies. Accessed: 2024-6-29. URL: https://github.com/phenoscape/owlery.

6 James P Balhoff. whelk-rs: Whelk is an OWL EL reasoner. Accessed: 2024-6-29. URL: https://github.com/INCATools/whelk-rs.

7 James P. Balhoff. Incatools/whelk-paper, October 2024. Software, v1.1.0. doi:10.5281/zenodo.13891879.

8 James P Balhoff and Christopher J Mungall. boomer: Bayesian OWL ontology merging. Accessed: 2024-6-29. URL: https://github.com/INCATools/boomer.

9 James P. Balhoff and Christopher J. Mungall. Whelk reasoner. Software, version 1.2.1., swhId: swh:1:dir:f8919707e053212b2c74bc63988a06ffe03fb796 (visited on 2024-12-10). URL: https://github.com/INCATools/whelk, doi:10.4230/artifacts.22615.

10 Katy Börner, Sarah A Teichmann, Ellen M Quardokus, James C Gee, Kristen Browne, David Osumi-Sutherland, Bruce W Herr, 2nd, Andreas Bueckle, Hrishikesh Paul, Muzlifah Haniffa, Laura Jardine, Amy Bernard, Song-Lin Ding, Jeremy A Miller, Shin Lin, Marc K Halushka, Avinash Boppana, Teri A Longacre, John Hickey, Yiing Lin, M Todd Valerius, Yongqun He, Gloria Pryhuber, Xin Sun, Marda Jorgensen, Andrea J Radtke, Clive Wasserfall, Fiona Ginty, Jonhan Ho, Joel Sunshine, Rebecca T Beuschel, Maigan Brusko, Sujin Lee, Rajeev Malhotra, Sanjay Jain, and Griffin Weber. Anatomical structures, cell types and biomarkers of the human reference atlas. *Nature cell biology*, 23(11):1117–1128, November 2021. doi:10.1038/s41556-021-00788-6.

11 Openllet code repository. openllet: Openllet is an OWL 2 reasoner in java, build on top of pellet. Accessed: 2024-6-29. URL: https://github.com/Galigator/openllet.

12 Alexander D Diehl, Terrence F Meehan, Yvonne M Bradford, Matthew H Brush, Wasila M Dahdul, David S Dougall, Yongqun He, David Osumi-Sutherland, Alan Ruttenberg, Sirarat Sarntivijai, Ceri E Van Slyke, Nicole A Vasilevsky, Melissa A Haendel, Judith A Blake, and Christopher J Mungall. The cell ontology 2016: enhanced content, modularization, and ontology interoperability. *Journal of biomedical semantics*, 7(1):44, July 2016. doi:10.1186/s13326-016-0088-7.

13 Robert B Doorenbos. *Production Matching for Large Learning Systems*. PhD thesis, Carnegie Mellon University, 1995. URL: http://reports-archive.adm.cs.cmu.edu/anon/1995/CMU-CS-95-113.pdf.

14 ELK code repository. Elk: Owl features. Accessed: 2024-10-04. URL: https://github.com/liveontologies/elk-reasoner/wiki/OwlFeatures.

15 Charles L Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, 19(1):17–37, September 1982. doi:10.1016/0004-3702(82)90020-0.

16 Michael A Gargano, Nicolas Matentzoglu, Ben Coleman, Eunice B Addo-Lartey, Anna V Anagnostopoulos, Joel Anderton, Paul Avillach, Anita M Bagley, Eduard Bakštein, James P Balhoff, Gareth Baynam, Susan M Bello, Michael Berk, Holli Bertram, Somer Bishop, Hannah Blau, David F Bodenstein, Pablo Botas, Kaan Boztug, Jolana Čady, Tiffany J Callahan, Rhiannon Cameron, Seth J Carbon, Francisco Castellanos, J Harry Caufield, Lauren E Chan, Christopher G Chute, Jaime Cruz-Rojo, Noémi Dahan-Oliel, Jon R Davids, Maud de Dieuleveult, Vinicius de Souza, Bert B A de Vries, Esther de Vries, J Raymond DePaulo, Beata Derfalvi, Ferdinand Dhombres, Claudia Diaz-Byrd, Alexander J M Dingemans, Bruno Donadille, Michael Duyzend, Reem Elfeky, Shahim Essaid, Carolina Fabrizzi, Giovanna Fico, Helen V Firth, Yun Freudenberg-Hua, Janice M Fullerton, Davera L Gabriel, Kimberly Gilmour, Jessica Giordano, Fernando S Goes, Rachel Gore Moses, Ian Green, Matthias Griese, Tudor Groza, Weihong Gu, Julia Guthrie, Benjamin Gyori, Ada Hamosh, Marc Hanauer, Kateřina Hanušová, Yongqun Oliver He, Harshad Hegde, Ingo Helbig, Kateřina Holasová, Charles Tapley Hoyt, Shangzhi Huang, Eric Hurwitz, Julius O B Jacobsen, Xiaofeng Jiang, Lisa Joseph, Kamyar Keramatian, Bryan King, Katrin Knoflach, David A Koolen, Megan L Kraus, Carlo Kroll, Maaike Kusters, Markus S Ladewig, David Lagorce, Meng-Chuan Lai, Pablo Lapunzina, Bryan Laraway, David Lewis-Smith, Xiarong Li, Caterina Lucano, Marzieh Majd, Mary L Marazita, Victor Martinez-Glez, Toby H McHenry, Melvin G McInnis, Julie A McMurry, Michaela Mihulová, Caitlin E Millett, Philip B Mitchell, Veronika Moslerová, Kenji Narutomi, Shahrzad Nematollahi, Julian Nevado, Andrew A Nierenberg, Nikola Novák Čajbiková, John I Nurnberger, Jr, Soichi Ogishima, Daniel Olson, Abigail Ortiz, Harry Pachajoa, Guiomar Perez de Nanclares, Amy Peters, Tim Putman, Christina K Rapp, Ana Rath, Justin Reese, Lauren Rekerle, Angharad M Roberts, Suzy Roy, Stephan J Sanders, Catharina Schuetz, Eva C Schulte, Thomas G Schulze, Martin Schwarz, Katie Scott, Dominik Seelow, Berthold Seitz, Yiping Shen, Morgan N Similuk, Eric S Simon, Balwinder Singh, Damian Smedley, Cynthia L Smith, Jake T Smolinsky, Sarah Sperry, Elizabeth Stafford, Ray Stefancsik, Robin Steinhaus, Rebecca Strawbridge, Jagadish Chandrabose Sundaramurthi, Polina Talapova, Jair A Tenorio Castano, Pavel Tesner, Rhys H Thomas, Audrey Thurm, Marek Turnovec, Marielle E van Gijn, Nicole A Vasilevsky, Markéta Vlčková, Anita Walden, Kai Wang, Ron Wapner, James S Ware, Addo A Wiafe, Samuel A Wiafe, Lisa D Wiggins, Andrew E Williams, Chen Wu, Margot J Wyrwoll, Hui Xiong, Nefize Yalin, Yasunori Yamamoto, Lakshmi N Yatham, Anastasia K Yocum, Allan H Young, Zafer Yüksel, Peter P Zandi, Andreas Zankl, Ignacio Zarante, Miroslav Zvolský, Sabrina Toro, Leigh C Car-

mody, Nomi L Harris, Monica C Munoz-Torres, Daniel Danis, Christopher J Mungall, Sebastian Köhler, Melissa A Haendel, and Peter N Robinson. The human phenotype ontology in 2024: phenotypes around the world. *Nucleic acids research*, 52(D1):D1333–D1346, January 2024. doi:10.1093/nar/gkad1005.

17 Gene Ontology Consortium, Suzi A Aleksander, James Balhoff, Seth Carbon, J Michael Cherry, Harold J Drabkin, Dustin Ebert, Marc Feuermann, Pascale Gaudet, Nomi L Harris, David P Hill, Raymond Lee, Huaiyu Mi, Sierra Moxon, Christopher J Mungall, Anushya Muruganugan, Tremayne Mushayahama, Paul W Sternberg, Paul D Thomas, Kimberly Van Auken, Jolene Ramsey, Deborah A Siegele, Rex L Chisholm, Petra Fey, Maria Cristina Aspromonte, Maria Victoria Nugnes, Federica Quaglia, Silvio Tosatto, Michelle Giglio, Suvarna Nadendla, Giulia Antonazzo, Helen Attrill, Gil Dos Santos, Steven Marygold, Victor Strelets, Christopher J Tabone, Jim Thurmond, Pinglei Zhou, Saadullah H Ahmed, Praoparn Asanitthong, Diana Luna Buitrago, Meltem N Erdol, Matthew C Gage, Mohamed Ali Kadhum, Kan Yan Chloe Li, Miao Long, Aleksandra Michalak, Angeline Pesala, Armalya Pritazahra, Shirin C C Saverimuttu, Renzhi Su, Kate E Thurlow, Ruth C Lovering, Colin Logie, Snezhana Oliferenko, Judith Blake, Karen Christie, Lori Corbani, Mary E Dolan, Harold J Drabkin, David P Hill, Li Ni, Dmitry Sitnikov, Cynthia Smith, Alayne Cuzick, James Seager, Laurel Cooper, Justin Elser, Pankaj Jaiswal, Parul Gupta, Pankaj Jaiswal, Sushma Naithani, Manuel Lera-Ramirez, Kim Rutherford, Valerie Wood, Jeffrey L De Pons, Melinda R Dwinell, G Thomas Hayman, Mary L Kaldunski, Anne E Kwitek, Stanley J F Laulederkind, Marek A Tutaj, Mahima Vedi, Shur-Jen Wang, Peter D'Eustachio, Lucila Aimo, Kristian Axelsen, Alan Bridge, Nevila Hyka-Nouspikel, Anne Morgat, Suzi A Aleksander, J Michael Cherry, Stacia R Engel, Kalpana Karra, Stuart R Miyasato, Robert S Nash, Marek S Skrzypek, Shuai Weng, Edith D Wong, Erika Bakker, Tanya Z Berardini, Leonore Reiser, Andrea Auchincloss, Kristian Axelsen, Ghislaine Argoud-Puy, Marie-Claude Blatter, Emmanuel Boutet, Lionel Breuza, Alan Bridge, Cristina Casals-Casas, Elisabeth Coudert, Anne Estreicher, Maria Livia Famiglietti, Marc Feuermann, Arnaud Gos, Nadine Gruaz-Gumowski, Chantal Hulo, Nevila Hyka-Nouspikel, Florence Jungo, Philippe Le Mercier, Damien Lieberherr, Patrick Masson, Anne Morgat, Ivo Pedruzzi, Lucille Pourcel, Sylvain Poux, Catherine Rivoire, Shyamala Sundaram, Alex Bateman, Emily Bowler-Barnett, Hema Bye-A-Jee, Paul Denny, Alexandr Ignatchenko, Rizwan Ishtiaq, Antonia Lock, Yvonne Lussi, Michele Magrane, Maria J Martin, Sandra Orchard, Pedro Raposo, Elena Speretta, Nidhi Tyagi, Kate Warner, Rossana Zaru, Alexander D Diehl, Raymond Lee, Juancarlos Chan, Stavros Diamantakis, Daniela Raciti, Magdalena Zarowiecki, Malcolm Fisher, Christina James-Zorn, Virgilio Ponferrada, Aaron Zorn, Sridhar Ramachandran, Leyla Ruzicka, and Monte Westerfield. The gene ontology knowledgebase in 2023. *Genetics*, 224(1), May 2023. doi:10.1093/genetics/iyad031.

18 Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: An OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, October 2014. doi:10.1007/s10817-014-9305-1.

19 Bruce W Herr, 2nd, Josef Hardi, Ellen M Quardokus, Andreas Bueckle, Lu Chen, Fusheng Wang, Anita R Caron, David Osumi-Sutherland, Mark A Musen, and Katy Börner. Specimen, biological structure, and spatial ontologies in support of a human reference atlas. *Scientific data*, 10(1):171, March 2023. doi:10.1038/s41597-023-01993-8.

20 Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011. doi:10.3233/SW-2011-0025.

21 Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. URL: https://www.w3.org/Submission/SWRL/.

22 SNOMED International. SNOMED international. Accessed: 2024-6-27. URL: https://www.snomed.org/.

23 Rebecca C Jackson, James P Balhoff, Eric Douglass, Nomi L Harris, Christopher J Mungall, and James A Overton. ROBOT: A tool for automating ontology workflows. *BMC bioinformatics*, 20(1):407, July 2019. doi:10.1186/s12859-019-3002-3.

24 Yevgeny Kazakov, Markus Krötzsch, and František Simančík. The incredible ELK. *Journal of Automated Reasoning*, 53(1):1–61, November 2013. doi:10.1007/s10817-013-9296-3.

25 Adila Krisnadhi, Frederick Maier, and Pascal Hitzler. OWL and rules. In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data: 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, pages 382–415. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-23032-5_7.

26 Phil Lord. horned-owl. Accessed: 2024-6-30. URL: https://github.com/phillord/horned-owl.

27 Nicolas Matentzoglu, Damien Goutte-Gattat, Shawn Zheng Kai Tan, James P Balhoff, Seth Carbon, Anita R Caron, William D Duncan, Joe E Flack, Melissa Haendel, Nomi L Harris, William R Hogan, Charles Tapley Hoyt, Rebecca C Jackson, Hyeongsik Kim, Huseyin Kir, Martin Larralde, Julie A McMurry, James A Overton, Bjoern Peters, Clare Pilgrim, Ray Stefancsik, Sofia Mc Robb, Sabrina Toro, Nicole A Vasilevsky, Ramona Walls, Christopher J Mungall, and David Osumi-Sutherland. Ontology development kit: a toolkit for building, maintaining and standardizing biomedical ontologies. *Database: the journal of biological databases and curation*, 2022, October 2022. doi:10.1093/database/baac087.

28 C J Mungall, H Dietze, and D Osumi-Sutherland. Use of OWL within the gene ontology. In Maria

Keet and Valentina Tamma, editors, *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED 2014)*, pages 25–36, Riva del Garda, Italy, October 17-18, 2014, October 2014. doi:10.1101/010090.

29 Christopher J Mungall, Sebastian Koehler, Peter Robinson, Ian Holmes, and Melissa Haendel. k-BOOM: A bayesian approach to ontology structure inference, with applications in disease ontology construction, May 2016. doi:10.1101/048843.

30 Christopher J Mungall, Carlo Torniai, Georgios V Gkoutos, Suzanna E Lewis, and Melissa A Haendel. Uberon, an integrative multi-species anatomy ontology. *Genome biology*, 13(1):R5, January 2012. doi:10.1186/gb-2012-13-1-r5.

31 Mark A Musen and Protégé Team. The protégé project: A look back and a look forward. *AI matters*, 1(4):4–12, June 2015. doi:10.1145/2757001.2757003.

32 Chris Okasaki. *Purely Functional Data Structures.* PhD thesis, Carnegie Mellon University, 1996. URL: https://www.cs.cmu.edu/~rwh/students/okasaki.pdf.

33 Tim E Putman, Kevin Schaper, Nicolas Matentzoglu, Vincent P Rubinetti, Faisal S Alquaddoomi, Corey Cox, J Harry Caufield, Glass Elsarboukh, Sarah Gehrke, Harshad Hegde, Justin T Reese, Ian Braun, Richard M Bruskiewich, Luca Cappelletti, Seth Carbon, Anita R Caron, Lauren E Chan, Christopher G Chute, Katherina G Cortes, Vinícius De Souza, Tommaso Fontana, Nomi L Harris, Emily L Hartley, Eric Hurwitz, Julius O B Jacobsen, Madan Krishnamurthy, Bryan J Laraway, James A McLaughlin, Julie A McMurry, Sierra A T Moxon, Kathleen R Mullen, Shawn T O'Neil, Kent A Shefchek, Ray Stefancsik, Sabrina Toro, Nicole A Vasilevsky, Ramona L Walls, Patricia L Whetzel, David Osumi-Sutherland, Damian Smedley, Peter N Robinson, Christopher J Mungall, Melissa A Haendel, and Monica C Munoz-Torres. The monarch initiative in 2024: an analytic platform integrating phenotypes, genes and diseases across species. *Nucleic acids research*, 52(D1):D938–D949, January 2024. doi:10.1093/nar/gkad1082.

34 Gunjan Singh, Sumit Bhatia, and Raghava Mutharaju. OWL2Bench: A benchmark for OWL 2 reasoners. In *The Semantic Web – ISWC 2020*, pages 81–96. Springer International Publishing, 2020. doi:10.1007/978-3-030-62466-8_6.

35 Nicholas Sioutos, Sherri de Coronado, Margaret W Haber, Frank W Hartel, Wen-Ling Shaiu, and Lawrence W Wright. NCI thesaurus: a semantic model integrating cancer-related clinical and molecular information. *Journal of biomedical informatics*, 40(1):30–43, February 2007. doi:10.1016/j.jbi.2006.02.013.

36 Barry Smith, Werner Ceusters, Bert Klagges, Jacob Köhler, Anand Kumar, Jane Lomax, Chris Mungall, Fabian Neuhaus, Alan L Rector, and Cornelius Rosse. Relations in biomedical ontologies. *Genome biology*, 6(5):R46, April 2005. doi:10.1186/gb-2005-6-5-r46.

37 Paul D Thomas, David P Hill, Huaiyu Mi, David Osumi-Sutherland, Kimberly Van Auken, Seth Carbon, James P Balhoff, Laurent-Philippe Albou, Benjamin Good, Pascale Gaudet, Suzanna E Lewis, and Christopher J Mungall. Gene ontology causal activity modeling (GO-CAM) moves beyond GO annotations to structured descriptions of biological functions and systems. *Nature genetics*, 51(10):1429–1433, October 2019. doi:10.1038/s41588-019-0500-1.

38 Santiago Timón-Reina, Mariano Rincón, Rafael Martínez-Tomás, Bjørn-Eivind Kirsebom, and Tormod Fladby. A knowledge graph framework for dementia research data. *NATO Advanced Science Institutes series E: Applied sciences*, 13(18):10497, September 2023. doi:10.3390/app131810497.

39 Nicole A Vasilevsky, Nicolas A Matentzoglu, Sabrina Toro, Joseph E Flack, IV, Harshad Hegde, Deepak R Unni, Gioconda F Alyea, Joanna S Amberger, Larry Babb, James P Balhoff, Taylor I Bingaman, Gully A Burns, Orion J Buske, Tiffany J Callahan, Leigh C Carmody, Paula Carrio Cordo, Lauren E Chan, George S Chang, Sean L Christiaens, Michel Dumontier, Laura E Failla, May J Flowers, H Alpha Garrett, Jr, Jennifer L Goldstein, Dylan Gration, Tudor Groza, Marc Hanauer, Nomi L Harris, Jason A Hilton, Daniel S Himmelstein, Charles Tapley Hoyt, Megan S Kane, Sebastian Köhler, David Lagorce, Abbe Lai, Martin Larralde, Antonia Lock, Irene López Santiago, Donna R Maglott, Adriana J Malheiro, Birgit H M Meldal, Monica C Munoz-Torres, Tristan H Nelson, Frank W Nicholas, David Ochoa, Daniel P Olson, Tudor I Oprea, David Osumi-Sutherland, Helen Parkinson, Zoë May Pendlington, Ana Rath, Heidi L Rehm, Lyubov Remennik, Erin R Riggs, Paola Roncaglia, Justyne E Ross, Marion F Shadbolt, Kent A Shefchek, Morgan N Similuk, Nicholas Sioutos, Damian Smedley, Rachel Sparks, Ray Stefancsik, Ralf Stephan, Andrea L Storm, Doron Stupp, Gregory S Stupp, Jagadish Chandrabose Sundaramurthi, Imke Tammen, Darin Tay, Courtney L Thaxton, Eloise Valasek, Jordi Valls-Margarit, Alex H Wagner, Danielle Welter, Patricia L Whetzel, Lori L Whiteman, Valerie Wood, Colleen H Xu, Andreas Zankl, Xingmin Aaron Zhang, Christopher G Chute, Peter N Robinson, Christopher J Mungall, Ada Hamosh, and Melissa A Haendel. Mondo: Unifying diseases for the world, by the world, April 2022. doi:10.1101/2022.04.13.22273750.

40 W3C OWL Working Group. Owl 2 web ontology language document overview (second edition). Accessed: 2024-10-04. URL: https://www.w3.org/TR/owl2-overview/.

41 W3C OWL Working Group. Owl 2 web ontology language manchester syntax (second edition). Accessed: 2024-10-04. URL: https://www.w3.org/TR/owl2-manchester-syntax/.

42 W3C OWL Working Group. OWL 2 web ontology language profiles (second edition). URL: https://www.w3.org/TR/owl2-profiles/.

43 École Polytechnique Fédérale Lausanne (EPFL). Scala native. Accessed: 2024-6-25. URL: https://scala-native.org/.

44 École Polytechnique Fédérale Lausanne (EPFL). The scala programming language. Accessed: 2024-6-30. URL: https://www.scala-lang.org/.

45 École Polytechnique Fédérale Lausanne (EPFL). Scala.js. Accessed: 2024-6-25. URL: https://www.scala-js.org.