YAGO 4: A Reason-able Knowledge Base

Thomas Pellissier-Tanon[1], Gerhard Weikum[2], and Fabian Suchanek[1]

[1] Télécom Paris, Institut Polytechnique de Paris
[2] Max Planck Institute for Informatics

Abstract. YAGO is one of the large knowledge bases in the Linked Open Data cloud. In this resource paper, we present its latest version, YAGO 4, which reconciles the rigorous typing and constraints of `schema.org` with the rich instance data of Wikidata. The resulting resource contains 2 billion type-consistent triples for 64 Million entities, and has a consistent ontology that allows semantic reasoning with OWL 2 description logics.

1 Introduction

A knowledge base (KB) is a machine-readable collection of knowledge about the real world. A KB contains entities (such as organizations, movies, people, and locations) and relations between them (such as *birthPlace*, *director*, etc.). KBs have wide applications in search engines, question answering, fact checking, chatbots, and many other NLP and AI tasks. Numerous projects have constructed KBs automatically or by help of a community. Notable KBs include YAGO [17], DBpedia [1], BabelNet [14], NELL [2], KnowItAll [3], and Wikidata [18]. On the industry side, giants such as Amazon, Google, Microsoft, Alibaba, Tencent and others are running KB technology as a background asset, often referred to as knowledge graphs.

YAGO [17,10,16,13] was one of the first academic projects to build a knowledge base automatically. The main idea of YAGO was to harvest information about entities from the infoboxes and categories of Wikipedia, and to combine this data with an ontological backbone derived from classes in WordNet [4]. Since Wikipedia is an excellent repository of entities, and WordNet is a widely used lexical resource, the combination proved useful. YAGO sent each fact through a pipeline of filtering, constraint checking, and de-duplication steps. This procedure scrutinized noisy input and boosted the quality of the final KB, to a manually verified accuracy of 95%. This precision was possible thanks to the tight control that the YAGO creators had over the extraction process, the filtering process, the ontological type system, the choice of the relations, and the semantic constraints. However, despite new versions YAGO2 and YAGO3 with substantial jumps in scope and size, the focus on Wikipedia infoboxes meant that YAGO has not arrived at the same scale as Freebase or Wikidata.

Meanwhile, Wikidata [18] has evolved into the world's foremost publicly available KB. It is a community effort where anybody can contribute facts – either

by manually adding or curating statements in the online interface, or by bulk-loading data. Wikidata has motivated more than 40,000 people who contribute at least once a month. The result is a public KB with 70M named entities, very good long-tail coverage, and impressive detail. [1]

At the same time, Wikidata understands itself as a collection of information, not as a collection of universally agreed-upon knowledge. It may intentionally contain contradictory statements, each with different sources or validity areas. Therefore, Wikidata does not enforce semantic constraints, such as "each person has exactly one father". Furthermore, the large user community has led to a proliferation of relations and classes: Wikidata contains 6.7k relations, of which only 2.6K have more than 1000 facts, and it comprises around 2.4M classes[2], of which 80% have less than 10 instances. Many instances (e.g., all cities) are placed in the taxonomy under more than 60 classes, with three-fold multiple inheritance. This complexity is the trade-off that Wikidata has found to accommodate its large user community. For downstream applications, the convoluted and often confusing type system of Wikidata make browsing and question answering tedious. Moreover, there is little hope to run strict classical reasoners (e.g., for OWL 2) in a meaningful way, as the KB contains many small inconsistencies so that every possible statement is deducible regardless of whether it is intuitively correct or false. Some of these issues have been pointed out in the comprehensive study of KB quality by [19].

Example. To illustrate the shortcomings by the verbose and sometimes confusing type hierarchy of Wikidata, consider the entities *Notre Dame de Paris* (`http://www.wikidata.org/entity/Q2981`) and *Potala Palace* (`http://www.wikidata.org/entity/Q71229`) both landmarks of two world religions.

Notre Dame is an instance of types *catholic cathedral* and *minor basilicas*, with a rich set of superclasses. The Potala Palace in Lhasa is an instance of *palace* and *tourist attraction*. Interestingly, the latter does not have Notre Dame de Paris as an instance, neither directly nor indirectly. So a query for tourist attractions would find the Potala Palace but not Notre Dame.

Moreover, the class *tourist attraction* is a subclass of *geographic object* which is an instance of the class *geometric concept* which in turn has superclass *mathematical concept*. As a consequence, a query for mathematical concepts returns entities like tensor, polynomial, differential equation ... and the *Potala Palace* as answers.

Contribution. In this resource paper, we describe the new YAGO version, YAGO 4, which aims to combine the best of the two worlds: It collects the facts about instances from Wikidata, but it forces them into a rigorous type hierarchy with semantic constraints. The complex taxonomy of Wikidata is replaced by the simpler and clean taxonomy of schema.org [8]. The classes are equipped

[1] All the numbers given in the paper about Wikidata are valid as of Feb. 24, 2020.

[2] Wikidata does not have a strong concept of a "class"; we use this term to denote entities that have superclasses (i.e., appear as left-hand argument of "subclass of" triples).

with SHACL constraints [12] that specify disjointness, applicable relations, and cardinalities. This way, YAGO 4 transfers the rationale of the original YAGO from the combination of Wikipedia and WordNet to the combination of Wikidata and schema.org. The result is a new knowledge base, which is not just large, but also logically consistent, so that OWL-based reasoning is feasible. Hence we call YAGO 4 a "reason-able" knowledge base. The new resource is available at `http://yago-knowledge.org` under a permissive license (Creative Commons Attribution-ShareAlike). YAGO 4 also comes with a browser and a SPARQL endpoint. Figure 1 shows an excerpt of the new YAGO in the online browser.

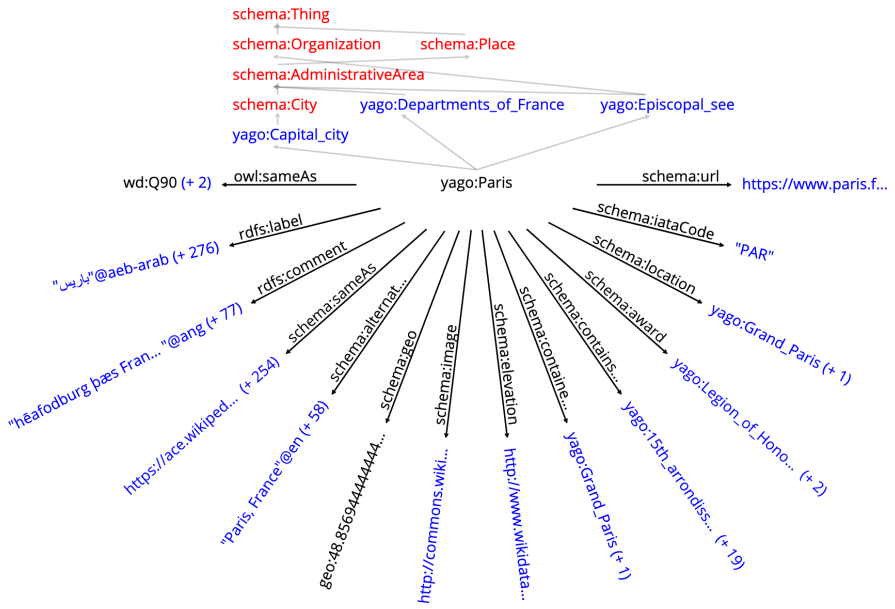


Fig. 1. The YAGO 4 Browser. Hovering reveals the full name of abbreviated items; all red and blue items are clickable.

2 Related Work

The Linked Open data cloud contains several dozen general-purpose KBs[3]. YAGO 4 is not intended to replace these KBs, but rather as an addition to this ecosystem with unique characteristics that complement the other players. For example, DBpedia also has a new version that ingests facts from Wikidata [11], with a well-designed pipeline that allows harvesting different knowledge sources [5]. This new DBpedia and YAGO 4 have made different design choices, resulting in different strengths and limitations. Our key priority has been to strengthen

[3] `https://www.lod-cloud.net/`

the logical rigor of the KB, so as to support OWL and other reasoners. This is why YAGO 4 builds on schema.org and adds its own constraint system which is much more elaborate than what DBpedia enforces.

3 Design

The construction of the YAGO 4 knowledge base is driven by several design decisions, which we explain and motivate next. The overarching point is to center YAGO 4 around a well-founded notion of classes. For example, a *Person* is defined as a subclass of *Thing*, and has an explicit set of possible relations such as *birthDate*, *affiliation*, etc.[4] Conversely, other relations such as *capitalOf*, *headquarter* or *population* are not applicable to instances of the class *Person*. This overarching principle of semantic consistency unfolds into several design choices.

3.1 Concise Taxonomy

Wikidata contains a very detailed taxonomy to which the community contributes by adding *instanceOf* and *subclassOf* statements. However, the resulting class hierarchy is so deep and convoluted that it is not easy to grasp and that browsing it is rather tedious. For example, Paris is an instance of 60 classes, 20 of which are called "unit", "entity", "subject", or "object". Moreover, the class hierarchy is not stable: any contributor can add or remove *subclassOf* links between any two classes. Potentially, this could lead to millions of entities being classified differently, just by a single edit. On the other hand, schema.org, the second major input to YAGO 4, has established itself as a reference taxonomy on the Web, beyond its initial aim at helping search engines to index web pages. It is stable, well maintained, and changes are made only by agreement in the W3C Schema.org Community Group[5]. At the same time, schema.org does not provide fine-grained classes such as "electric cars" or "villages" – which only Wikidata has. Schema.org also does not have any biochemical classes (such as proteins etc.).

We address the latter problem by using *Bioschemas* [7][6]. This project extends schema.org in the field of the life sciences – a field that is not covered in schema.org, and that is very prominent in Wikidata. We manually merged 6 Bioschemas classes into schema.org, referring to the merged taxonomy as the "schema.org taxonomy" for simplicity.

To obtain the stability of schema.org while preserving the fine-grained classes of Wikidata, we found the following solution: The top-level taxonomy of YAGO 4 is taken from schema.org (incl. Bioschemas), and leaf-level classes are taken from Wikidata. For this purpose, we manually mapped 235 classes of schema.org to Wikidata classes. Classes of schema.org that could not be mapped, mostly

[4] For readability, we omit namespace prefixes in this paper.
[5] `https://www.w3.org/community/schemaorg/`
[6] `https://bioschemas.org`

shopping-related or social-media classes such as *schema:LikeAction*, were removed. With these inputs, the YAGO 4 taxonomy is then constructed as follows:

- For each instance in Wikidata, we consider each possible path in the Wikidata taxonomy to the root node. If the first class on the path has a Wikipedia article, we include it in YAGO 4. The rationale is that only classes with an English Wikipedia article are of sufficient interest for a wider audience and use cases.
- We then continue the path to the root in the Wikidata taxonomy, discarding all classes on the way, until we hit a class that has been mapped to schema.org. We continue our path to the root in the schema.org taxonomy, adding all classes on the way to YAGO 4.
- If we do not hit a class that has been mapped to schema.org, we discard the entire path. If an instance has no path with a class that qualifies for these criteria, we discard the instance.

We discard all Wikidata classes that have less than 10 direct instances. This threshold serves to ignore classes that have little value in use cases or are rather exotic. We further remove subclasses of a small list of meta-level Wikidata classes such as Wikipedia categories, disambiguation pages, etc. Finally, we drop subclasses of pair of classes for which we enforce disjointness constraints. These design choices allow us to model villages and cars, while significantly reducing the size of the taxonomy. From the 2.4M original Wikidata classes, we kept only 10k classes, shrinking the taxonomy by 99.6%. We also discard 11M instances (14%) – two thirds of which (7.5M) are Wikipedia-specific meta-entities (disambiguation page, category, wikitext template, etc.). Our strategy capitalizes on the stable backbone of schema.org, while being able to augment YAGO 4 with new data coming from Wikidata.

3.2 Legible Entities and Relations

YAGO 4 is stored in the RDF format. Unlike Wikidata, we chose to give human-readable URIs to all entities, in order to make the KB more accessible for interactive use. If an entity has a Wikipedia page (which we know because Wikidata links it to Wikipedia), we take the Wikipedia title as the entity name. Otherwise, we concatenate the English label of the entity with its Wikidata identifier (e.g., *Bischmisheim_Q866094*). Studies like [15] suggest that the Wikidata labels are fairly stable, leading to fairly stable YAGO URIs. If the entity has no English label, we stay with the Wikidata identifier. We make the necessary changes to arrive at a valid local IRI name, and add the namespace of YAGO, `http://yago-knowledge.org/resource/`. This gives the vast majority of entities human-readable names, without introducing duplicates or ambiguity.

Wikidata has a very rich set of relations, but many of these have only very few facts. Indeed 61% of them have less than 1000 facts and 85% of them less than 10k. For YAGO 4, we chose to follow the successful model of previous YAGO versions, which have been parsimonious on the relations per class. We chose the

relations from schema.org, which are each attached to a class. While these relations are conservative in coverage, they have emerged as a useful reference. We mapped 116 of these relations manually to the relations of Wikidata. We simply add this information to our schema, by using two new relations, *yago:fromClass* and *yago:fromProperty*, as shown here:

schema:Person yago:fromClass wd:Q215627
yago:birthPlaceProperty yago:fromProperty wdt:P569

The pipeline for KB construction takes care to implement these mappings (Section 4.1). This process discards around 7k relations from Wikidata. As a by-product, it gives human-readable names to all relations. Example relations are *schema:birthPlace, schema:founder*, and *schema:containedInPlace*. We use RDF and RDFS relations whenever possible, including *rdfs:label* and *rdfs:comment* instead of *schema:name* and *schema:description*. For example, the fact "wd:Q42 wdt:P31 wd:Q5" from Wikidata becomes

yago:Douglas_Adams rdf:type schema:Person

3.3 Well-typed Values

YAGO 4 has not just well-typed entities, but also well-typed literals. For this purpose, we translate the data values of Wikidata to RDF terms. References to Wikidata entities are converted to references to the YAGO entities as explained in Section 3.2. External URIs are converted into *xsd:anyURI* literals after normalizing them.[7] We chose to keep external URIs as literals and not as entities, because we do not make any statements about URIs. Time values are converted to *xsd:dateTime*, *xsd:date*, *xsd:gYearMonth* or *xsd:gYear*, depending on the time precision. We discard the other time values whose precision could not be mapped to an XML schema type. Globe coordinates are mapped to *schema:GeoCoordinates* resources. Quantities are mapped to *schema:QuantitativeValue* resources (keeping the unit and precision). If there is no unit and an empty precision range, we map to *xsd:integer* where possible. If the unit is a duration unit (minutes, seconds...) and the precision range is empty, we map to *xsd:duration*. In this way, the vast majority of values are migrated to standard RDF typed literals.

3.4 Semantic Constraints

YAGO 4 has hand-crafted semantic constraints that not just keep the data clean, but also allow logical reasoning on the data. We model constraints in the W3C standards SHACL [12] and OWL. YAGO 4 currently has the following constraints:

[7] We follow the normalization suggested by RFC 2986 Section 6.2.

Disjointness. We specify 6 major top-level classes: *schema:BioChemicalEntity*, *schema:Event*, *schema:Organization*, *schema:Person*, *schema:Place*, and *schema:CreativeWork*. With the exception of *schema:Organization*/*schema:Place*, these are pairwise disjoint; so that these classes cannot have any instances in common. We use OWL to express, for example:

schema:Person owl:disjointWith schema:CreativeWork

Note that organizations are not disjoint from places, because many organizations are also located somewhere.

Domain and Range. Each relation comes with a domain and range constraint, meaning that a relation such as *birthPlace* can apply only to a person and a place. RDFS can specify the domain and range of relations by help of the predicates *rdfs:domain* and *rdfs:range*, but our constraints are different: If a KB contained the fact *birthPlace(London, Paris)*, then the statement *rdfs:domain(birthPlace,Person)* would simply deduce that London must be a person. In contrast, our constraints would flag the KB as inconsistent. We use SHACL to express these constraints, as in this example:

schema:Person sh:property yago:birthPlaceProperty
yago:birthPlaceProperty sh:path schema:birthPlace
yago:birthPlaceProperty sh:node schema:Place

The same property can be used to describe entities of different classes. For example *telephone* can be used to describe both persons and organizations. In this case, the same property is going to be in the shapes of several classes. The domain of the property then is the union of all these classes.

In the same spirit, we also support disjunction in property ranges. For example, the range of *author* is *Person* union *Organization*. Following the same argument, the range of the *birthDate* property is the union of datatypes *xsd:dateTime*, *xsd:date*, *xsd:gYearMonth* and *xsd:gYear* to allow different calendar value precisions. Our range constraints also include the validation of *xsd:string* literals via regular expressions, as in this example:

schema:Person sh:property yago:telephoneProperty
yago:telephoneProperty sh:path schema:telephone
yago:telephoneProperty sh:pattern "+\d{1,3} ..."

Functional Constraints. A functional constraint says that a relation can have at most one object for a subject. Several of our relations are functional, e.g., *birthPlace* or *gender*. Again, we use SHACL:

yago:Person sh:property yago:birthPlaceProperty
yago:birthPlaceProperty sh:maxCount "1"^^xsd:integer

Cardinality Constraints. Going beyond functional constraints, we can also specify the maximal number of objects in general. For example, people can have only two parents in YAGO 4. We use again the SHACL *sh:maxCount* property.

YAGO 4 assumes that no other properties are allowed for each class, thereby interpreting the SHACL constraints under a "closed world assumption". The constraints are automatically enforced during the construction of the KB (see Section 4.1), and so the data of YAGO 4 satisfies all constraints. Overall, the enforcement of constraints leads to the removal of 132M facts from Wikidata (i.e. 28% of all the facts). Since the constraints are enforced at KB-construction time, we can then add the deductive *rdfs:domain* and *rdfs:range* facts to YAGO 4 without risking that these deduce anything that violates the constraints.

The generated ontology uses the OWL 2 axioms *DisjointClasses*, *ObjectPropertyDomain*, *DataPropertyDomain*, *ObjectPropertyRange*, *DataPropertyRange*, *ObjectUnionOf*, *FunctionalDataProperty*, *FunctionalObjectProperty*, and falls into the OWL DL flavor. Statistics about the mapping and constraints are shown in Table 1.

Table 1. Schema and mapping statistics

Item	Number
Schema.org classes	235
Bioschemas.org classes	6
Object properties	100
Datatype properties	41
Node shapes	49
Property shapes	217
Domain constraints	217
Object range constraints	132
Datatype range constraints	57
Regex constraints	21
Disjoint constraints	18

3.5 Annotations for Temporal Scope

Following previous YAGO versions, YAGO 4 also attaches temporal information to its facts. We harvest these from the Wikidata qualifier system, which annotates facts with their validity time, provenance, and other meta information. We express the temporal scopes of facts by the relations *schema:startDate* and *schema:endDate*. Instead of relying on a custom format for these annotations, we made use of the RDF* model proposal [9], which has received good traction in recent years. For example, we state that Douglas Adams lived in Santa Barbara until 2001 as follows:

<< Douglas_Adams schema:homeLocation Santa_Barbara >> schema:endDate 2001

We cannot use the usual Property Graph (PG) semantics of RDF*, because this would assert that Douglas Adams still lives in Santa Barbara. Rather, we use the "separate-assertions mode" (SA mode), which asserts only that he lived in Santa Barbara until 2001 – without saying where he currently lives.

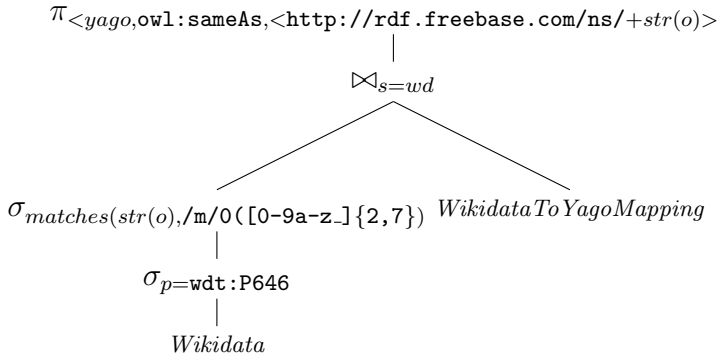
4 Knowledge Base

4.1 Construction

We have designed a system that builds YAGO 4 automatically from (1) a Wikidata dump and (2) the SHACL shapes definitions of Section 3. We keep only the "truthy" Wikidata statements, i.e. for each subject and predicate we keep only the statements with the "best" rank (a.k.a. "preferred" if a statement with such a rank exists, "normal" if not).

The KB building system constructs the class hierarchy, the entities, and the facts as outlined in Section 3. Its main purpose is then to enforce the constraints (Section 3.4). If a resource is an instance of disjoint classes, we drop the two *rdf:type* relations leading to this conflict. We drop all instances that are not instances of any class. We enforce domain, range and regular-expression constraints by pruning all candidate facts that would violate a constraint. Finally, we check the cardinality constraints, removing all objects if there are too many for a given subject.

Our system is implemented in the Rust programming language[8], using the `Iterator` infrastructure to ingest and output data streams. We use the already existing stream operators, which resemble those of relational algebra (map/project, filter, flat map, collect/materialize into a hash structure). We also implemented new operators particularly for YAGO 4 (stream-hash join, stream-hash anti join, group-by, and transitive closure). For example, the *owl:sameAs* links between YAGO 4 and Freebase can be extracted from Wikidata by the following algebraic operator plan:



Here, π is the projection operator, σ the selection, $\bowtie$ the inner join, $Wikidata$ the table of all Wikidata triples (s, p, o), and $WikidataToYagoMapping$ the mapping between Wikidata and YAGO instances $(wd, yago)$. To avoid reading the full Wikidata N-Triples dump each time, we first load the Wikidata dump into the RocksDB key-value store to index its content[9]. This index allows for efficiently selecting triples based on a predicate or a (predicate, subject) tuple, and getting back a stream of triples from the database.

[8] `https://www.rust-lang.org/`
[9] `https://rocksdb.org/`

The advantage of having operator plans in Rust is that we can benefit from declarative programs where performance optimizations are carried out by the compiler, generating highly efficient native code. After having loaded the data into RocksDB, our execution plan generates the Wikipedia-flavored YAGO 4 (see below) in two hours on a commodity server.

We ran our system on a dump of 78M Wikidata items. 8M of these are entities about Wikimedia Websites-related entities, such as categories. From the 474M Wikidata facts whose property has been mapped to schema.org, we filtered out 89M of them because of the domain constraints and 42M more because of the range and regex constraints. The cardinality constraints lead to the removal of an extra 0.6M facts.

4.2 Data

YAGO 4 is made available in three "flavors":

- **Full:** This flavor uses all data from Wikidata, resulting in a very large KB.
- **Wikipedia:** This smaller flavor of YAGO 4 contains only the instances that have a Wikipedia article (in any language).
- **English Wikipedia:** This is an additional restriction of the Wikipedia flavor, containing only instances that have an English Wikipedia article.

All three flavors of YAGO 4 are built in the same way, and have the same schema, with 116 properties and the same taxonomy of 140 top-level classes from schema.org and bioschemas.org, and the same subset of Wikidata classes. Table 2 shows statistics for the three YAGO 4 variants, generated from the Wikidata N-Triples dump of November 25, 2019.

Each flavor of YAGO 4 is split into the following files:

- **Taxonomy:** The full taxonomy of classes.
- **Full-types:** All *rdf:type* relations.
- **Labels:** All entity labels (*rdfs:label*, *rdfs:comment* and *schema:alternateName*).
- **Facts:** The facts that are not labels.
- **Annotations:** The fact annotations encoded in RDF* [9].
- **SameAs:** The *owl:sameAs* links to Wikidata, DBpedia, and Freebase and the *schema:sameAs* to all the Wikipedias.
- **Schema:** The schema.org classes and properties, in OWL 2 DL.
- **Shapes:** The SHACL constraints used to generate YAGO 4.

Each file is a compressed N-Triples file, so that standard tools can directly ingest the data.

Table 2. Size statistics for YAGO 4 in the flavors Full, Wikipedia (W), and English Wikipedia (E), Wikidata and DBpedia (per DBpedia SPARQL server on 2020-03-04).

	Yago Full	Yago W	Yago E	Wikidata	DBpedia
Classes	10124	10124	10124	2.4M	484k
Classes from Wikidata	9883	9883	9883	2.4M	222
Individuals	67M	15M	5M	78M	5M
Labels (*rdfs:label*)	303M	137M	66M	371M	22M
Descriptions (*rdfs:comment*)	1399M	139M	50M	2146M	12M
Aliases (*schema:alternateName*)	68M	21M	14M	71M	0
rdf:type (without transitive closure)	70M	16M	5M	77M	114M
Facts	343M	48M	20M	974M	131M
Avg. # of facts per entity	5.1	3.2	4	12.5	26
sameAs to Wikidata	67M	15M	5M	N.A.	816k
sameAs to DBpedia	5M	5M	5M	0	N.A.
sameAs to Freebase	1M	1M	1M	1M	157k
sameAs to Wikipedia	43M	43M	26M	66M	13M
Fact annotations	2.5M	2.2M	1.7M	220M	0
Dump size	60GB	7GB	3GB	127GB	99GB

4.3 Access

Web Page. The YAGO 4 knowledge base is available at `http://yago-knowledge.org`. The Web page offers an introduction to YAGO, documentation ("Getting started"), and a list of publications and contributors. The Web page also has a schema diagram that lists all top-level classes with their associated relations and constraints.

License. The entire YAGO 4 knowledge base, as well as all previous versions and the logo, can be downloaded from the Web page. YAGO 4 is available under a Creative Commons Attribution-ShareAlike License. The reason for this choice is that, while Wikidata is in the public domain, schema.org is under a Creative Commons Attribution-ShareAlike License.[10]

Source Code. We have released the source code for constructing YAGO 4 on GitHub at `https://github.com/yago-naga/yago4` under the GNU GPL v3+ license.

SPARQL Endpoint. YAGO 4 comes with a responsive SPARQL endpoint, which can be used as an API or interactively. The URL is `http://yago-knowledge.org/sparql/query`. The YAGO URIs are also all dereferencable, thus complying with the Semantic Web best practice.

Browser. YAGO 4 comes with a graphical KB browser, with an example shown in Figure 1. For each entity, the browser visualizes the outgoing relationships in a

[10] `http://schema.org/docs/terms.html`

star-shape around the entity. Above the entity, the browser shows the hierarchy of all classes of which the entity is a (transitive) instance, including those with multiple inheritance. If an entity has more than one object for a given relation, a relation-specific screen shows all objects of that relation for the entity. For size reasons, the browser shows only the Wikipedia flavor of YAGO.

Applications. YAGO has already been used in quite a number of projects [16], including question answering, entity recognition, and semantic text analysis. We believe that the new version of YAGO opens up the door to an entire array of new applications, because it is possible to perform logical reasoning on YAGO 4. Not only is the KB equipped with semantic constraints, but it is also provably consistent. We have checked the "English Wikipedia" flavor of YAGO 4 with the OWL 2 DL reasoner HermiT [6], proving its logical consistency.[11] This makes it possible to perform advanced kinds of logical inference on YAGO 4.

5 Conclusion

This paper presents YAGO 4, the newest version of the YAGO knowledge base. The unique characteristics of YAGO 4 is to combine the wealth of facts from Wikidata with the clean and human-readable taxonomy from schema.org, together with semantic constraints that enforce logical consistency. This way, the resulting KB can be processed with OWL and other reasoners, and is also more user-friendly for browsing and question answering. We hope that the YAGO 4 resource fills a gap in the landscape of public KBs, and will be useful in downstream applications.

We plan to release updates of YAGO 4 to reflect the changes in Wikidata. A change of the schema vocabulary would require human intervention, and could be done a few times a year. Future work includes extending the set of semantic constraints to capture inverse functions, symmetric and transitive properties, and more. We also consider tapping into additional data sources, beyond Wikidata, to further enrich the factual knowledge of YAGO 4.

Acknowledgements. This work was partially supported by the grant ANR-16-CE23-0007-01 ("DICOS").

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: Dbpedia: A nucleus for a web of open data. In: ISWC. pp. 722–735 (2007). https://doi.org/10.1007/978-3-540-76298-0_52
2. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E.R.H., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: AAAI (2010), `http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1879`

[11] HermiT was unable to load the "Full" flavor due to a memory overflow, but it contains the same taxonomy and the same constraints as the "English Wikipedia" flavor.

3. Etzioni, O., Cafarella, M.J., Downey, D., Kok, S., Popescu, A., Shaked, T., Soderland, S., Weld, D.S., Yates, A.: Web-scale information extraction in knowitall. In: WWW. pp. 100–110 (2004). https://doi.org/10.1145/988672.988687
4. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press (1998), https://mitpress.mit.edu/books/wordnet
5. Frey, J., Hofer, M., Obraczka, D., Lehmann, J., Hellmann, S.: DBpedia FlexiFusion the best of Wikipedia > Wikidata > your data. In: ISWC. pp. 96–112 (2019). https://doi.org/10.1007/978-3-030-30796-7_7
6. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An OWL 2 reasoner. J. Autom. Reasoning **53**(3), 245–269 (2014). https://doi.org/10.1007/s10817-014-9305-1
7. Gray, A.J.G., Goble, C.A., Jimenez, R.: Bioschemas: From potato salad to protein annotation. In: ISWC (2017), http://ceur-ws.org/Vol-1963/paper579.pdf
8. Guha, R.V., Brickley, D., Macbeth, S.: Schema.org: evolution of structured data on the web. Commun. ACM **59**(2), 44–51 (2016). https://doi.org/10.1145/2844544
9. Hartig, O.: Foundations of RDF⋆ and SPARQL⋆ (an alternative approach to statement-level metadata in RDF). In: Alberto Mendelzon Workshop on Foundations of Data Management and the Web (2017), http://ceur-ws.org/Vol-1912/paper12.pdf
10. Hoffart, J., Suchanek, F.M., Berberich, K., Lewis-Kelham, E., de Melo, G., Weikum, G.: YAGO2: exploring and querying world knowledge in time, space, context, and many languages. In: WWW. pp. 229–232 (2011). https://doi.org/10.1145/1963192.1963296
11. Ismayilov, A., Kontokostas, D., Auer, S., Lehmann, J., Hellmann, S.: Wikidata through the eyes of dbpedia. Semantic Web **9**(4), 493–503 (2018). https://doi.org/10.3233/SW-170277
12. Knublauch, H., Kontokostas, D.: Shapes constraint language (SHACL). W3C Candidate Recommendation **11**(8) (2017), https://www.w3.org/TR/shacl/
13. Mahdisoltani, F., Biega, J., Suchanek, F.M.: YAGO3: A knowledge base from multilingual wikipedias. In: CIDR (2015), http://cidrdb.org/cidr2015/Papers/CIDR15_Paper1.pdf
14. Navigli, R., Ponzetto, S.P.: Babelnet: Building a very large multilingual semantic network. In: ACL. pp. 216–225 (2010), https://www.aclweb.org/anthology/P10-1023/
15. Pellissier Tanon, T., Kaffee, L.: Property label stability in wikidata: Evolution and convergence of schemas in collaborative knowledge bases. In: WikiWorkshop, WWW. pp. 1801–1803 (2018). https://doi.org/10.1145/3184558.3191643
16. Rebele, T., Suchanek, F.M., Hoffart, J., Biega, J., Kuzey, E., Weikum, G.: YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames. In: ISWC. pp. 177–185 (2016). https://doi.org/10.1007/978-3-319-46547-0_19
17. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: WWW. pp. 697–706 (2007). https://doi.org/10.1145/1242572.1242667
18. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014). https://doi.org/10.1145/2629489
19. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment for linked data: A survey. Semantic Web **7**(1), 63–93 (2016). https://doi.org/10.3233/SW-150175