Inside YAGO2s: A Transparent Information Extraction Architecture

Joanna Biega
Max Planck Institute
for Informatics
Saarbrücken, Germany
jbiega@mpi-inf.mpg.de

Erdal Kuzey
Max Planck Institute
for Informatics
Saarbrücken, Germany
ekuzey@mpi-inf.mpg.de

Fabian M. Suchanek
Max Planck Institute
for Informatics
Saarbrücken, Germany
suchanek@mpi-inf.mpg.de

ABSTRACT

YAGO[9, 6] is one of the largest public ontologies constructed by information extraction. In a recent refactoring called YAGO2s, the system has been given a modular and completely transparent architecture. In this demo, users can see how more than 30 individual modules of YAGO work in parallel to extract facts, to check facts for their correctness, to deduce facts, and to merge facts from different sources. A GUI allows users to play with different input files, to trace the provenance of individual facts to their sources, to change deduction rules, and to run individual extractors. Users can see step by step how the extractors work together to combine the individual facts to the coherent whole of the YAGO ontology.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms

Keywords

YAGO, Information Extraction, Ontologies

1. INTRODUCTION

Recent years have seen a rise of large-scale information extraction projects: NELL[3], PROSPERA [8], SOFIE [10], and TextRunner[2] extract facts from natural language Web documents, DBpedia [1] and YAGO [9, 6] extract from Wikipedia pages, and several commercial endeavors have started exploring the area, too. TrueKnowledge[1], Freebase[2], and Google's Knowledge Graph, among others, all extract information from Web sources. These systems usually take Web pages as input, and produce a knowledge base (KB) as output. The KB is a set of facts, such as e.g., "Elvis Presley was born in Tupelo". The facts are usually represented in the RDF format or in a similar format, meaning that every fact is a triple. A triple is a statement of a subject, a predicate, and an object. In the example, the triple is ⟨*Elvis_Presley, bornIn, Tupelo*⟩. Today's KBs contain millions of such triples about a wide variety of entities. They know, e.g., which cities are located in which countries, which actors played in which movies, and which scientists won which award.

While some knowledge bases extract their data from arbitrary Web pages, the YAGO project extracts its data exclusively from a handfull of predefined sources. YAGO also works only on a manually predefined set of relations. In return, the facts in YAGO have a very high precision. A manual evaluation[9, 6] has confirmed that 95% of the triples in YAGO are correct.

Achieving this correctness is no simple task, because YAGO draws from few, but very different sources. The system extracts and merges information from Wikipedia, WordNet, Geonames, the Universal WordNet, and WordNet Domains. Facts have to be extracted from the Infoboxes, the categories, and the full text of Wikipedia, and reconciled with conflicting, duplicate, or complementary facts from the other sources. Entities have to be mapped and deduplicated, and class hierarchies have to be merged and combined. In addition, a suite of verifications has to be applied to make sure that domain- and range-constraints are respected, that functional relations have no more than one object for any given subject, and that the types of an entity are coherent with each other. This entire process takes several days to run. Furthermore, the YAGO team has steadily grown, which requires a careful distribution of responsibilities. Apart from this, more than a dozen researchers at our institute work directly or indirectly on the knowledge base.

To adapt to these conditions, we have taken a radical step, and refactored the YAGO system architecture from scratch. The new system is a transparent and modular architecture of over 30 independent extraction systems, each of which is handled by one or two responsibles. This way, our architecture allows us to collaboratively develop our knowledge base. The extraction systems work in parallel to generate the YAGO ontology. Even though the extractors are independent, the overall architecture is designed in such a way that it enforces the consistency and uniqueness of the facts. Thus, even though the facts are extracted from different sources by different systems, the result is a coherent and consistent whole.

With the present demo proposal, we would like to show this architecture in a playful and interactive fashion. Our purpose is two-fold: First, we believe that the architecture,

[1]`http://trueknowledge.com`
[2]`http://freebase.com`

as well as many of its components, can inspire other information extraction projects that face the same challenges. Second, we would also like to receive feedback and proposals for improvement of our own system from the demo visitors. For these reasons, we have developed a novel, graphical, interactive version of the YAGO system. Users can play with the system "in vivo", try it out on toy examples, inspect or modify inputs or outputs of the systems, show fact provenance, and run individual components. We believe that this experience could provide inspiration for both the visitors and our own team.

2. THE YAGO2S ARCHITECTURE

YAGO. The YAGO[9] system extracts information from several sources. The titles of articles in Wikipedia constitute the entities of YAGO. The categories of Wikipedia articles are analyzed to derive the type of the entities. The infoboxes are parsed to extract facts about the entities. WordNet[5] delivers the taxonomic backbone of the ontology. With YAGO2 [6], more geographic entities were merged into YAGO from the Geonames gazetteer[3]. This version also gave the ontology a spatial and a temporal dimension. YAGO also became multilingual by help of class labels from the Universal WordNet (UWN)[4]. YAGO uses its own knowledge to check the consistency of newly extracted facts. All facts are automatically checked with respect to the type signature of their relations, and with respect to functionality constraints. Both facts and entities are deduplicated. All entities are required to have at least one type, and the system enforces the types of an entity to be compatible with each other.

YAGO2s. We have completely refactored the YAGO system into a transparent and modular architecture. The refactored version of YAGO is called YAGO2s. The main ingredients of the new architecture are *themes* and *extractors*. A theme is a collection of facts, such as all facts extracted from Wikipedia infoboxes, all facts derived from WordNet, or all facts that concern people. A theme is stored in a file. We took the opportunity to make the syntax of YAGO facts completely RDF compliant, and store facts in the Turtle format. In order to attach time and space information to facts, YAGO uses fact identifiers [6]. In the Turtle format, we store these fact identifiers in commented lines before the facts. This allows advanced applications to deal with temporal and spatial information, while still allowing standard RDF applications to load and consume the files.

Extractors. An extractor is a module of code, which takes a number of themes as input, and produces a number of themes as output. For example, one extractor is the *deduplicator*, which takes a number of themes as input, and produces one theme with the deduplicated facts as output. Other extractors check types, verify functional constraints, or merge information. Some extractors also extract information from an external data source. These extractors take a raw data file as an additional input. The *Wikipedia category extractor*, e.g., takes as input the XML dump of Wikipedia and produces a theme with facts extracted from Wikipedia categories. Similar extractors exist for WordNet, UWN, and Geonames. We also added an extractor for WordNet domains[7]. The WordNet domains give YAGO a thematic structure of topics, such as "music", "law", and "emotions". Therefore, it is now possible to ask for all entities related to, e.g., "music".

Scheduling. An extractor can only be run once its input themes have been produced. This constraint yields a dependency graph, in which some extractors have to run before others, and some can run in parallel. We have designed a scheduler that respects these dependencies. Of the 30 extractors, up to 14 run in parallel, producing around 80 themes in 4 days on a 8-core machine. The interplay of data extractors and verifying extractors ensures that all facts that make it into the final layer of the architecture have been checked for consistency and uniqueness. Together, the themes of the final layer constitute the YAGO ontology. These themes group the facts thematically, so that facts about multilingual labels, facts about literals, and facts about inter-ontology links, e.g., all appear in different themes. The final themes can be downloaded separately, so that users can download just what they need ("YAGO à la carte").

3. THE YAGO2S DEMO

3.1 GUI

Visualization. Our demo proposal visualizes the YAGO architecture as a bipartite graph of themes and extractors. The graph is dynamically created from the actual registered extractors. Figure 1 shows the entire graph in miniature to give an impression of the structure of the graph. The circles are extractors, the boxes are themes, and the smaller boxes are data sources. The lines show which extractor produces which theme, and which theme is used by which extractors. This graph constitutes our GUI. Users can zoom in and out to discover different types of extractors and their dependencies. Figure 2 shows a zoom on one particular extractor, the Wikipedia Type Extractor.

Extractors. There are 4 main groups of extractors: *Wikipedia Extractors* are concerned exclusively with Wikipedia. All of them take Wikipedia as an external data source. *Geonames Extractors* harvest and map entities from Geonames. *External Extractors* extract from external sources other than Wikipedia and Geonames, such as WordNet, WordNet Domains, and UWN. *Theme Extractors* take no external data sources, and operate just on themes. These extractors deduplicate, merge, and check constraints. Some extractors (such as the Type Checker) are instantiated multiple times, so that the overall graph contains 46 extractors. Themes are not replicated. They exist only once and are accessed in parallel by the extractors that consume them.

Implementation. The architecture graph is represented in the Scalable Vector Format (SVG), so that it can be conveniently displayed and zoomed in a browser. A static version of this image is also available online at `http://mpi-inf.mpg.de/yago/demo`. The backend system is programmed in Java, with the interactive part running as a local TomCat server and the dynamic interface written in JavaScript.

[3]`http://www.geonames.org`

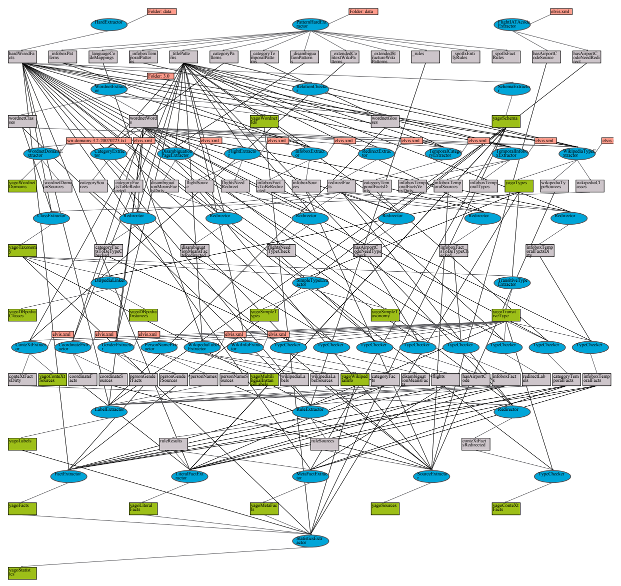

Figure 1: The dependency graph of YAGO2s

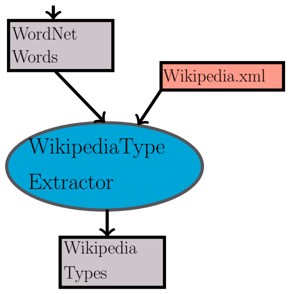

Figure 2: A zoom on Figure 1

3.2 Basic Functionalities

Our GUI not only visualizes the architecture, but also lets the user interact with it. Each theme, and each extractor has a context menu. These support the following operations:

Preview Themes. The user can choose to explore the facts contained in a theme. Since YAGO contains general-purpose facts, the themes can be easily understood. Some contain person names, others contain geographic information, and others again contain multilingual labels for entities or classes.

Display Theme Dependencies. The user can highlight all extractors that use a certain theme. Some themes are used by only one extractor, whereas others have more intricate dependencies. Some themes are used by more than a dozen extractors. There is one theme, *yagoSchema*, which defines the schema of the ontology. It contains the names of the relations, their domains and ranges, and the top-level classes of YAGO. This theme is used by virtually all extractors. This ensures that the entire system operates on the same relation definitions.

Display Extractor Dependencies. The user can highlight the subgraph of elements (extractors and themes) that depend on a certain extractor. For example, facts produced by the *Infobox Extractor* are first checked for redirects, then verified by the type checker, then fed into a rule deduction mechanism, and finally merged with the other facts and deduplicated.

Display Extractor Input. The user can highlight all input themes and data files required by an extractor. Due to complex dependencies, these may be quite scattered in the graph.

Run Extractors. The user can decide to run an individual extractor. Since we plan to operate mostly on toy data during the demo, each extractor will run in a few seconds. This allows for a tight loop of interaction where the user can modify the input and run chosen extractors to see how facts trickle through the system or get intercepted.

Modify Data Sources. One can preview data sources that the system shall run on, inspect them, and even modify them. For instance, the user can choose to run the Wikipedia extractor only on a specific Wikipedia page. The user can modify the information on the Wikipedia page or introduce inconsistent data to see how the system handles it.

Modify Themes. The same flexibility is available for the themes. The user can modify intermediate themes and see how this affects the ouput. Of particular interest are the themes that define the schema of YAGO, the extraction patterns for Wikipedia, and the themes that define the deductive rules of YAGO. A reset button will always return the system to its pristine state.

Display Provenance. If users wish to know how a certain fact emerged, they can ask the system for the provenance of this fact. This will display the path of the fact from the final themes through the checkers and verifiers up to the data sources. Some facts can come from multiple sources, yielding a tree in the visualization. Others are filtered out on the way.

3.3 Demo Scenarios

Our running example is Ronaldo, the footballer from Rio de Janeiro. We will run our extractors on his Wikipedia page only. We will focus on facts about the prizes he got during his football career (relation *hasWonPrize*).

Simple Extraction. As a first example, we propose to run the Category Extractor. It will find 3 *hasWonPrize* facts in the categories of Ronaldo's Wikipedia page:

⟨*Ronaldo, hasWonPrize, European Footballer of the Year*⟩
⟨*Ronaldo, hasWonPrize, FIFA World Player of the Year*⟩
⟨*Ronaldo, hasWonPrize, World Soccer Magazine Player*⟩

The next extractor in the line is the Redirector. It takes care of Wikipedia redirects, so that facts will have canonical entities. In our example, the entity *European Footballer of the Year* will be redirected to the entity *Ballon d'Or*. The next extractor to be run is the type checker. The type checker validates the facts by the domain and range constraints of the relation. In our example, the type of the entity *World Soccer Magazine Player* is not known to YAGO. Hence, the type checker will eliminate this fact. Thus, the result of the process is

⟨*Ronaldo, hasWonPrize, Ballon d'Or*⟩
⟨*Ronaldo, hasWonPrize, FIFA World Player of the Year*⟩

Editing the input. The user can edit the Wikipedia page and introduce new facts. In our running example, the user can add the following line to the infobox of the Wikipedia page: "*awards=[[European Golden Shoe]]*" When the Infobox Extractor is run, its output will contain the fact ⟨*Ronaldo, hasWonPrize, European Golden Shoe*⟩.

Tricking the system. The user can also introduce noisy, inconsistent or wrong data on the page, and see whether the system is robust enough to deal with it. The user can, e.g., add the following line to the infobox:

"*awards= [[European Golden Shoe]], [[FIFA]], 2002*"

The Infobox Extractor will not extract *2002* as an award, because it is of the wrong syntactic type. However, it will assume that *FIFA* is an award that Ronaldo won, ⟨*Ronaldo, hasWonPrize, FIFA*⟩. The subsequent Type Checker will determine that *FIFA* is not an award, and it will eliminate this fact, so that only the *European Golden Shoe* remains as an award.

The user can also add inconsistent categories to the page. For example, the user can state that Ronaldo is in the category *Cities in Italy*. The Type Extractor will detect the anomality of this category: It is not a subclass of *person*, like all the other categories of Ronaldo. Thus, the Type Extractor will not extract ⟨*Ronaldo, rdf:type, city*⟩. However, the Category Extractor will be tricked into assuming that Ronaldo is located in Italy, ⟨*Ronaldo, locatedIn, Italy*⟩. The subsequent Type Checker will remark and remove that fact.

Changing the YAGO schema. Another interesting interaction is to modify the themes that define the schema of YAGO. If the user declares, e.g., a relation as a function, then the system will extract at most one fact of that relation per subject. To illustrate, if the non-functional relation *hasWonPrize* is changed to be functional, then only the fact ⟨*Ronaldo, hasWonPrize, Ballon d'Or*⟩ will be extracted. Moreover, the user can add a totally new relation to the schema, together with the domain, range, and the type of the relation. If she wants to see the contests that Ronaldo won, she can simply add the new relation *hasWonContest*. She should define the domain as *person*, and the range as *contest*. Then, if there is any extraction rule associated with this relation, the system will extract facts about contest winners.

Editing the extraction rules. The extractors use pattern-based rules to extract information from Wikipedia pages. Our GUI allows users to edit or add new extraction rules. For instance, if a user wants to extract facts about contest winners, then she can add the following extraction rule on Wikipedia categories

(.+)–winning players ⟶ ⟨*$0, hasWonContest, $1*⟩

This rule says that if an article has a category that matches the regular expression on the left hand side of the rule, then the system will generate a fact of the form ⟨*$0, hasWonContest, $1*⟩ (where $0 is the article entity, and $1 is the first group of the regular expression). In the example of Ronaldo's Wikipedia page, the rule will match the category "*FIFA Confederations Cup–winning players*", and thus generate the fact ⟨*Ronaldo, hasWonContest, FIFA Confederations Cup*⟩.

Editing the induction rules. The YAGO system implements a light-weight deduction mechanism over the facts. For example, YAGO already has the relation *participatedIn* that indicates that an entity participated in an event or activity. In the original YAGO, there is no such fact about Ronaldo. However, if the user introduced the relation *hasWonContest*, then we can create an induction rule saying "*If someone wins a contest, then he also participated in that contest*". The rule in YAGO format will be like

⟨*$0, hasWonPrize, $1*⟩ ⇒ ⟨*$0, participatedIn, $1*⟩

If the user runs the Rule Extractor, this will induce the fact ⟨*Ronaldo, participatedIn, FIFA Confederations Cup*⟩. We provide a video of our demo at `http://mpi-inf.mpg.de/yago/demo`.

4. CONCLUSION

With this paper, we propose to demonstrate the new architecture of the YAGO extraction system. The user can follow step by step how our system transforms the semi-structured input of Wikipedia into fact candidates, and then into clean and consistent RDF facts. YAGO is an example of an extensible knowledge extraction system, where multiple researchers collaborate to develop a knowledge base. We believe that the architecture can inspire other researchers in the area of information extraction by providing ideas for the organization and building blocks of ontology construction. We anticipate that the demo would be of particular interest to researchers in the area of database provenance, because YAGO is a system that uses and generates provenance information. We also hope that this demo will encourage visitors to give feedback on our system, to point out weaknesses of the extractors, and to propose new extractors. This will allow us to make YAGO ever more useful.

5. REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A nucleus for a Web of open data. In *ISWC*, 2007.

[2] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, 2007.

[3] A. Carlson, J. Betteridge, R. C. Wang, E. R. H. Jr., and T. M. Mitchell. Coupled semi-supervised learning for information extraction. In *WSDM*, 2010.

[4] G. de Melo and G. Weikum. Towards a universal wordnet by learning from combined evidence. In *CIKM*, 2009.

[5] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[6] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence Journal*, 2013.

[7] B. Magnini and G. Cavaglia. Integrating subject field codes into wordnet. In *LREC*, 2000.

[8] N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *WSDM*, 2011.

[9] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge. Unifying WordNet and Wikipedia. In *WWW*, 2007.

[10] F. M. Suchanek, M. Sozio, and G. Weikum. SOFIE: A Self-Organizing Framework for Information Extraction. In *WWW*, 2009.