Yago: A Core of Semantic Knowledge

Fabian M. Suchanek, Gjergji Kasneci and Gerhard Weikum

MPI–I–2006–5-006 November 2006

Authors' Addresses

Fabian M. Suchanek
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Gjergji Kasneci
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Gerhard Weikum
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Abstract

We present YAGO, a light-weight and extensible ontology with high coverage and quality. YAGO builds on entities and relations and currently contains roughly 900,000 entities and 5,000,000 facts. This includes the Is-A hierarchy as well as non-taxonomic relations between entities (such as HASWONPRIZE). The facts have been automatically extracted from the unification of Wikipedia and WordNet, using a carefully designed combination of rule-based and heuristic methods described in this paper. The resulting knowledge base is a major step beyond WordNet: in *quality* by adding knowledge about individuals like persons, organizations, products, etc. with their semantic relationships – and in *quantity* by increasing the number of facts by more than an order of magnitude. Our empirical evaluation of fact correctness shows an accuracy of about 95%. YAGO is based on a logically clean model, which is decidable, extensible, and compatible with RDFS. Finally, we show how YAGO can be further extended by state-of-the-art information extraction techniques.

Keywords

Ontologies, Knowledge Extraction, Wikipedia, WordNet

Contents

1 Introduction

1.1 Motivation

Many applications in modern information technology utilize ontological background knowledge. This applies above all to applications in the vision of the Semantic Web, but there are many other application fields. Machine translation (e.g. [5]) and word sense disambiguation (e.g. [3]) exploit lexical knowledge, query expansion uses taxonomies (e.g. [15, 11, 27]), document classification based on supervised or semi-supervised learning can be combined with ontologies (e.g. [14]), and [13] demonstrates the utility of background knowledge for question answering and information retrieval. Furthermore, ontological knowledge structures play an important role in data cleaning (e.g., for a data warehouse) [6], record linkage (aka. entity resolution) [7], and information integration in general [18].

But the existing applications typically use only a single source of background knowledge (mostly WordNet [10] or Wikipedia). They could boost their performance, if a huge ontology with knowledge from several sources was available. Such an ontology would have to be of high quality, with accuracy close to 100 percent, i.e. comparable in quality to an encyclopedia. It would have to comprise not only concepts in the style of WordNet, but also named entities like people, organizations, geographic locations, books, songs, products, etc., and also relations among these such as what-is-located-where, who-was-born-when, who-has-won-which-prize, etc. It would have to be extensible, easily re-usable, and application-independent. If such an ontology were available, it could boost the performance of existing applications and also open up the path towards new applications in the Semantic Web era.

1.2 Related Work

Knowledge representation is an old field in AI and has provided numerous models from frames and KL-ONE to recent variants of description logics and

RDFS and OWL (see [21] and [23]). Numerous approaches have been proposed to create general-purpose ontologies on top of these representations. One class of approaches focusses on extracting knowledge structures automatically from text corpora. These approaches use information extraction technologies that include pattern matching, natural-language parsing, and statistical learning [25, 9, 4, 1, 22, 19, 8]. These techniques have also been used to extend WordNet by Wikipedia individuals [20]. Another project along these lines is KnowItAll [9], which aims at extracting and compiling instances of unary and binary predicate instances on a very large scale – e.g., as many soccer players as possible or almost all company/CEO pairs from the business world. Although these approaches have recently improved the quality of their results considerably, the quality is still significantly below that of a man-made knowledge base. Typical results contain many false positives (e.g., IsA(Aachen Cathedral, City), to give one example from KnowItAll). Furthermore, obtaining a recall above 90 percent for a closed domain typically entails a drastic loss of precision in return. Thus, information-extraction approaches are only of little use for applications that need near-perfect ontologies (e.g. for automated reasoning). Furthermore, they typically do not have an explicit (logic-based) knowledge representation model.

Due to the quality bottleneck, the most successful and widely employed ontologies are still man-made. These include WordNet [10], Cyc or OpenCyc [16], SUMO [17], and especially domain-specific ontologies and taxonomies such as SNOMED[1] or the GeneOntology[2]. These knowledge sources have the advantage of satisfying the highest quality expectations, because they are manually assembled. However, they suffer from low coverage, high cost for assembly and quality assurance, and fast aging. No human-made ontology knows the most recent Windows version or the latest soccer stars.

1.3 Contributions and Outline

This paper presents YAGO[3], a new ontology that combines high coverage with high quality. Its core is assembled from one of the most comprehensive lexicons available today, Wikipedia. But rather than using information extraction methods to leverage the knowledge of Wikipedia, our approach utilizes the fact that Wikipedia has *category pages*. Category pages are lists of articles that belong to a specific category (e.g., Zidane is in the category of French football players[4]). These lists give us candidates for entities

[1]http://www.snomed.org

[2]http://www.geneontology.org/

[3]Yet Another Great Ontology

[4]Soccer is called football in some countries

(e.g. Zidane), candidates for concepts (e.g. IsA(Zidane, FootballPlayer)) and candidates for relations (e.g. isCitizenOf(Zidane, France)). In an ontology, concepts have to be arranged in a taxonomy to be of use. The Wikipedia categories are indeed arranged in a hierarchy, but this hierarchy is barely useful for ontological purposes. For example, Zidane is in the super-category named "Football in France", but Zidane is a football *player* and not a football. WordNet, in contrast, provides a clean and carefully assembled hierarchy of thousands of concepts. But the Wikipedia concepts have no obvious counterparts in WordNet.

In this paper we present new techniques that link the two sources with near-perfect accuracy. To the best of our knowledge, our method is the first approach that accomplishes this unification between WordNet and facts derived from Wikipedia with an accuracy of 97%. This allows the YAGO ontology to profit, on one hand, from the vast amount of individuals known to Wikipedia, while exploiting, on the other hand, the clean taxonomy of concepts from WordNet. Currently, YAGO contains roughly 900,000 entities and 5 million relations between them.

YAGO is based on a data model of entities and binary relations. But by means of reification (i.e., introducing identifiers for relation instances) we can also express relations between relation instances (e.g., popularity rankings of pairs of soccer players and their teams) and general properties of relations (e.g., transitivity or acyclicity). We show that, despite its expressiveness, the YAGO data model is decidable.

YAGO is designed to be extendable by other sources – be it by other high quality sources (such as gazetteers of geographic places and their relations), by domain-specific extensions, or by data gathered through information extraction from Web pages. We conduct an enrichment experiment with the state-of-the-art information extraction system LEILA[25]. We show that the more facts YAGO contains, the better it can be extended. We observe that this positive feedback loop could even accelerate future extensions.

The rest of this paper is organized as follows. In Chapter 2 we introduce YAGO's data model. Chapter 3 describes the sources from which the current YAGO is assembled, namely, Wikipedia and WordNet. In Chapter 4 we give an overview of the system behind YAGO. We explain our extraction techniques and we show how YAGO can be extended by new data. Chapter 5 presents an evaluation, a comparison to other ontologies, an enrichment experiment and sample facts from YAGO. We conclude with a summary in Chapter 6.

2 The YAGO model

2.1 Structure

To accommodate the ontological data we already extracted and to be prepared for future extensions, YAGO must be based on a thorough and expressive data model. The model must be able to express entities, facts, relations between facts and properties of relations. The state-of-the-art formalism in knowledge representation is currently the Web Ontology Language OWL [23]. Its most expressive variant, OWL-full, can express properties of relations, but is undecidable. RDFS, the basis of OWL, can also express properties of relations, but provides only very primitive semantics (e.g. it does not know transitivity). This is why we introduce a slight extension of RDFS, the *YAGO model*. The YAGO model can express relations between facts and relations, while it is at the same time decidable and computationally simple.

As in OWL and RDFS, all beings (e.g. cities, people, even URLs) are represented as *entities* in the YAGO model. Two entities can stand in a *relation*. For example, to state that Albert Einstein won the Nobel Prize, we say that the entity `Albert Einstein` stands in the HASWONPRIZE relation with the entity `Nobel Prize`. We write

`AlbertEinstein` HASWONPRIZE `NobelPrize`

Numbers, dates, strings and other literals are represented as entities as well. This means that they can stand in relations to other entities. For example, to state that Albert Einstein was born in 1879, we write:

`AlbertEinstein` BORNINYEAR `1879`

Entities are abstract ontological beings, which are language-independent in the ideal case. Language uses words to refer to these entities. In the YAGO model, words are entities as well. This makes it possible to express that a certain word refers to a certain entity, like in the following example:

"*Einstein*" MEANS `AlbertEinstein`

This allows us to deal with synonymy and ambiguity. The following line says that "Einstein" may also refer to the musicologist Alfred Einstein:

"*Einstein*" MEANS `AlfredEinstein`

We use quotes to distinguish words from other entities. Similar entities are grouped into *classes*. For example, the class `physicist` comprises all physicists and the class `word` comprises all words. Each entity is an *instance* of at least one class. We express this by the TYPE relation:

`AlbertEinstein` TYPE `physicist`

Classes are also entities. Thus, each class is itself an instance of a class, namely of the class `class`. Classes are arranged in a taxonomic hierarchy, expressed by the SUBCLASSOF relation:

`physicist` SUBCLASSOF `scientist`

In the YAGO model, relations are entities as well. This makes it possible to represent properties of relations (like transitivity or subsumption) within the model. The following line, e.g., states that the SUBCLASSOF relation is transitive by making it an instance of the class `transitiveRelation`:

`subclassOf` TYPE `transitiveRelation`

The triple of an entity, a relation and an entity is called a *fact*. The two entities are called the *arguments* of the fact. Each fact is given a *fact identifier*. As RDFS, the YAGO model considers fact identifiers to be entities as well. This allows us to represent for example that a certain fact was found at a certain URL. For example, suppose that the above fact (`Albert Einstein`, BORNINYEAR, `1879`) had the fact identifier `#1`, then the following line would say that this fact was found in Wikipedia:

#1 FOUNDIN `http://www.wikipedia.org/Einstein`

We will refer to entities that are neither facts nor relations as *common entities*. Common entities that are not classes will be called *individuals*. Then, a YAGO ontology over a finite set of common entities $\mathcal{C}$, a finite set of relation names $\mathcal{R}$ and a finite set of fact identifiers $\mathcal{I}$ is a function

$$y : \mathcal{I} \rightarrow (\mathcal{I} \cup \mathcal{C} \cup \mathcal{R}) \times \mathcal{R} \times (\mathcal{I} \cup \mathcal{C} \cup \mathcal{R})$$

A YAGO ontology y has to be injective and total to ensure that every fact identifier of $\mathcal{I}$ is mapped to exactly one fact.

Some facts require more than two arguments (for example the fact that Einstein won the Nobel Prize in 1921). One common way to deal with this problem is to use n-ary relations (as for example in `won-prize-in-year(Einstein, Nobel-Prize, 1921)`). In a relational database setting, where relations correspond to tables, this has the disadvantage that much space will be wasted if not all arguments of the n-ary facts are known. Worse, if an argument (like e.g. the place of an event) has not been foreseen in the design phase of the database, the argument cannot be represented. Another way of dealing with an n-ary relation is to introduce a binary relation for each argument (e.g. `winner`, `prize`, `time`). Then, an n-ary fact can be represented by a new entity that is linked by these binary relations to all of its arguments (as it is proposed for OWL):

`AlbertEinstein` WINNER `EinsteinWonNP1921`
`NobelPrize` PRIZE `EinsteinWonNP1921`
`1921` TIME `EinsteinWonNP1921`

However, this method cannot deal with additional arguments for relations that were designed to be binary. The YAGO model offers a simple solution to this problem: It is based on the assumption that for each n-ary relation, a *primary pair* of its arguments can be identified. For example, for the above `won-prize-in-year`-relation, the pair of the person and the prize could be considered a primary pair. The primary pair can be represented as a binary fact with a fact identifier:

#1 : `AlbertEinstein` HASWONPRIZE `NobelPrize`

All other arguments can be represented as relations that hold between the primary pair and the other argument:

#2 : #1 TIME `1921`

2.2 Semantics

This section will give a model-theoretic semantics to YAGO. We first prescribe that the set of relation names $\mathcal{R}$ for any YAGO ontology must contain at least the relation names `type`, `subClassOf`, `domain`, `range` and `subRelationOf`. The set of common entities $\mathcal{C}$ must contain at least the classes `entity`, `class`, `relation`, `acyclicTransitiveRelation` and classes for all literals (as evident from the following list). For the rest of the paper, we assume a given set of common entities $\mathcal{C}$ and a given set of relations $\mathcal{R}$. The set of fact identifiers used by a YAGO ontology y is implicitly given by $\mathcal{I} = domain(y)$. To define the semantics of a YAGO ontology, we consider only the set of possible facts $\mathcal{F} = (\mathcal{I} \cup \mathcal{C} \cup \mathcal{R}) \times \mathcal{R} \times (\mathcal{I} \cup \mathcal{C} \cup \mathcal{R})$. We define a rewrite system $\rightarrow \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{P}(\mathcal{F})$, i.e. $\rightarrow$ reduces one set of facts to another set of facts. We use the shorthand notation $\{f_1, ..., f_n\} \hookrightarrow f$ to say that

$$F \cup \{f_1, ..., f_n\} \rightarrow F \cup \{f_1, ..., f_n\} \cup \{f\}$$

for all $F \subseteq \mathcal{F}$, i.e. if a set of facts contains the facts $f_1, ..., f_n$, then the rewrite rule adds f to this set. Our rewrite system contains the following (*axiomatic*) rules:[1]

$\emptyset \hookrightarrow$ (`domain`, DOMAIN, `relation`)
$\emptyset \hookrightarrow$ (`domain`, RANGE, `class`)
$\emptyset \hookrightarrow$ (`range`, DOMAIN, `relation`)
$\emptyset \hookrightarrow$ (`range`, RANGE, `class`)
$\emptyset \hookrightarrow$ (`subClassOf`, TYPE, `acyclicTransitiveRelation`)
$\emptyset \hookrightarrow$ (`subClassOf`, DOMAIN, `class`)
$\emptyset \hookrightarrow$ (`subClassOf`, RANGE, `class`)
$\emptyset \hookrightarrow$ (`type`, RANGE, `class`)
$\emptyset \hookrightarrow$ (`subRelationOf`, TYPE, `acyclicTransitiveRelation`)
$\emptyset \hookrightarrow$ (`subRelationOf`, DOMAIN, `relation`)
$\emptyset \hookrightarrow$ (`subRelationOf`, RANGE, `relation`)
$\emptyset \hookrightarrow$ (`boolean`, SUBCLASSOF, `literal`)
$\emptyset \hookrightarrow$ (`number`, SUBCLASSOF, `literal`)
$\emptyset \hookrightarrow$ (`rationalNumber`, SUBCLASSOF, `number`)
$\emptyset \hookrightarrow$ (`integer`, SUBCLASSOF, `rationalNumber`)
$\emptyset \hookrightarrow$ (`timeInterval`, SUBCLASSOF, `literal`)
$\emptyset \hookrightarrow$ (`dateTime`, SUBCLASSOF, `timeInterval`)
$\emptyset \hookrightarrow$ (`date`, SUBCLASSOF, `timeInterval`)
$\emptyset \hookrightarrow$ (`string`, SUBCLASSOF, `literal`)
$\emptyset \hookrightarrow$ (`character`, SUBCLASSOF, `string`)
$\emptyset \hookrightarrow$ (`word`, SUBCLASSOF, `string`)
$\emptyset \hookrightarrow$ (`URL`, SUBCLASSOF, `string`)

Furthermore, it contains the following rules for all $r, r_1, r_2 \in \mathcal{R}$, $x, y, c, c_1, c_2 \in \mathcal{I} \cup \mathcal{C} \cup \mathcal{R}$, $r_1 \neq$ TYPE, $r_2 \neq$ SUBRELATIONOF, $r \neq$ SUBRELATIONOF, $r \neq$ TYPE, $c \neq$ `acyclicTransitiveRelation`, $c_2 \neq$ `acyclicTransitiveRelation`:

(1) $\{(r_1, \text{SUBRELATIONOF}, r_2), (x, r_1, y)\} \hookrightarrow (x, r_2, y)$
(2) $\{(r, \text{TYPE}, \texttt{acyclicTransitiveRelation}), (x, r, y), (y, r, z)\}$
$\hookrightarrow (x, r, z)$
(3) $\{(r, \text{DOMAIN}, c), (x, r, c)\} \hookrightarrow (x, \text{TYPE}, c)$

[1]The class hierarchy of literals is inspired by SUMO[17]

(4) $\{(r, \textsc{range}, c), (x, r, y)\} \hookrightarrow (y, \textsc{type}, c)$
(5) $\{(x, \textsc{type}, c_1), (c_1, \textsc{subClassOf}, c_2)\} \hookrightarrow (x, \textsc{type}, c_2)$

THEOREM 1: [*Convergence of* $\rightarrow$]
Given a set of facts $F \subset \mathcal{F}$, *the largest set* S *with* $F \rightarrow^* S$ *is unique.*

(The theorems are proven in the appendix.) Given a YAGO ontology y, the rules of $\rightarrow$ can be applied to its set of facts, $range(y)$. We call the largest set that can be produced by applying the rules of $\rightarrow$ the *set of derivable facts* of y, $D(y)$. Two YAGO ontologies y_1, y_2 are *equivalent* if the fact identifiers in y_2 can be renamed so that

$$(y_1 \subseteq y_2 \quad \vee \quad y_2 \subseteq y_1) \quad \wedge \quad D(y_1) = D(y_2)$$

The *deductive closure* of a YAGO ontology y is computed by adding the derivable facts to y. Each derivable fact (x, r, y) needs a new fact identifier, which is just $f_{x,r,y}$. Using a relational notation for the function y, we can write this as

$$\begin{aligned} y^* \;:=\; y \;\cup\; \{\; & (f_{r,x,y}, (r, x, y)) \;\mid \\ & (x, r, y) \in D(y)\;,\; (r, x, y) \notin range(y) \;\} \end{aligned}$$

A *structure* for a YAGO ontology y is a triple of

- a set $\mathcal{U}$ (the *universe*)
- a function $\mathcal{D} : \mathcal{I} \cup \mathcal{C} \cup \mathcal{R} \rightarrow \mathcal{U}$ (the *denotation*)
- a function $\mathcal{E} :\; \mathcal{D}(\mathcal{R}) \rightarrow \mathcal{U} \times \mathcal{U}$ (the *extension function*)

Like in RDFS, a YAGO structure needs to define the extensions of the relations by the extension function $\mathcal{E}$. $\mathcal{E}$ maps the denotation of a relation symbol to a relation on universe elements. We define the *interpretation* Ψ with respect to a structure $< \mathcal{U}, \mathcal{D}, \mathcal{E} >$ as the following relation:

$$\Psi := \{(e_1, r, e_2) \;\mid\; (\mathcal{D}(e_1), \mathcal{D}(e_2)) \in \mathcal{E}(\mathcal{D}(r))\}$$

We say that a fact (e_1, r, e_2) is *true* in a structure, if it is contained in the interpretation. A *model* of a YAGO ontology y is a structure such that

1. all facts of y^* are true
2. if $\Psi(x, \textsc{type}, \texttt{literal})$ for some x, then $\mathcal{D}(x) = x$
3. if $\Psi(r, \textsc{type}, \texttt{acyclicTransitiveRelation})$ for some r, then there exists no x such that $\Psi(x, r, x)$

A YAGO ontology y is called *consistent* iff there exists a model for it. Obviously, a YAGO ontology is consistent iff

$$\begin{aligned}\nexists x, r : \ & (r, \textsc{type}, \texttt{acyclicTransitiveRelation}) \in D(y) \\ & \wedge \ (x, r, x) \in D(y)\end{aligned}$$

Since $D(y)$ is finite, the consistency of a YAGO ontology is decidable. A *base* of a YAGO ontology y is any equivalent YAGO ontology b with $b \subseteq y$. A *canonical base* of y is a base so that there exists no other base with less elements.

THEOREM 2: [*Uniqueness of the Canonical Base*]
The canonical base of a consistent YAGO ontology is unique.

In fact, the canonical base of a YAGO ontology can be computed by greedily removing facts from the ontology. This makes the canonical base a natural choice to efficiently store a YAGO ontology.

2.3 Relation to Other Formalisms

The YAGO model is very similar to RDFS. In RDFS, relations are called *properties*. Just as YAGO, RDFS knows the properties `domain`, `range`, `subClassOf` and `subPropertyOf` (i.e. `subRelationOf`). These properties have a semantics that is equivalent to that of the corresponding YAGO relations. RDFS also knows fact identifiers, which can occur as arguments of other facts. The following excerpt shows how some sample facts of Chapter 2.1 can be represented in RDFS. Each fact of YAGO becomes a triple in RDFS.

```
<rdf:Description
  rdf:about="http://mpii.mpg.de/yago#Albert_Einstein">
  <yago:bornIn rdf:ID="f1">1879</yago:bornIn>
</rdf:Description>
<rdf:Description
  rdf:about="http://mpii.mpg.de/yago#f1">
  <yago:foundIn rdf:ID="f2" rdf:resource="http:..."/>
</rdf:Description>
```

However, RDFS does not have a built-in transitive relation or an acyclic transitive relation, as YAGO does. This entails that the property `acyclic-TransitiveRelation` can be defined and used, but that RDFS would not know its semantics.

YAGO uses fact identifiers, but it does not have built-in relations to make logical assertions about facts (e.g. it does not allow to say that a fact is false). If one relies on the denotation to map a fact identifier to the corresponding fact element in the universe, one can consider fact identifiers as simple individuals. This abandons the syntactic link between a fact identifier and the fact. In return, it opens up the possibility of mapping a YAGO ontology to an OWL ontology under certain conditions. OWL has built-in counterparts for almost all built-in data types, classes, and relations of YAGO. The only concept that does not have an exact built-in counterpart is the `acyclicTransitiveRelation`. However, this is about to change. OWL is currently being refined to its successor, OWL 1.1. The extended description logic $\mathcal{SROIQ}$ [12], which has been adopted as the logical basis of OWL 1.1, allows to express irreflexivity and transitivity. This allows to define acyclic transitivity. We plan to investigate the relation of YAGO and OWL once OWL 1.1 has been fully established.

3 Sources for YAGO

3.1 WordNet

WordNet is a semantic lexicon for the English language developed at the Cognitive Science Laboratory of Princeton University. WordNet distinguishes between words as literally appearing in texts and the actual senses of the words. A set of words that share one sense is called a *synset*. Thus, each synset identifies one sense (i.e., semantic concept). Words with multiple meanings (ambiguous words) belong to multiple synsets. As of the current version 2.1, WordNet contains 81,426 synsets for 117,097 unique nouns. (Wordnet also includes other types of words like verbs and adjectives, but we consider only nouns in this paper.) WordNet provides relations between synsets such as hypernymy/hyponymy (i.e., the relation between a sub-concept and a super-concept) and holonymy/meronymy (i.e., the relation between a part and the whole); for this paper, we focus on hypernyms/hyponyms. Conceptually, the hypernymy relation in WordNet spans a directed acyclic graph (DAG) with a single source node called `Entity`.

3.2 Wikipedia

Wikipedia is a multilingual, Web-based encyclopedia. It is written collaboratively by volunteers and is available for free. We downloaded the English version of Wikipedia in August 2006, which comprised 1,200,000 articles at that time. Each Wikipedia article is a single Web page and usually describes a single topic.

The majority of Wikipedia pages have been manually assigned to one or multiple *categories*. The page about Albert Einstein, for example, is in the categories `German language philosophers`, `Swiss physicists`, and 34 more. Conveniently, the categorization of Wikipedia pages and their link structure are available as SQL tables, so that they can be exploited without parsing the actual Wikipedia articles.

4 The YAGO system

Our system is designed to extract a YAGO ontology from WordNet and Wikipedia. Currently, the relations of YAGO are fixed. Their properties (such as DOMAIN and RANGE) are described in Table 5.2.

YAGO is designed to be extendable, i.e. new facts from new sources can be added to the ontology. For this purpose, each fact is tagged with a confidence value. In the current YAGO, all facts have a confidence of 1.0, but facts extracted by other techniques (e.g. based on statistical learning) can have smaller confidence values.

4.1 Knowledge Extraction

4.1.1 The type relation

Since Wikipedia knows far more individuals than WordNet, the individuals for YAGO are taken from Wikipedia. Each Wikipedia page title is a candidate to become an individual in YAGO. For example, the page title "Albert Einstein" is a candidate to become the individual `AlbertEinstein` in our ontology. The page titles in Wikipedia are unique.

The Wikipedia Category System. To establish for each individual its class, we exploit the category system of Wikipedia. There are different types of categories: Some categories, the *conceptual categories*, indeed identify a class for the entity of the page (e.g. Albert Einstein is in the category *Naturalized citizens of the United States*). Other categories serve administrative purposes (e.g. Albert Einstein is also in the category *Articles with unsourced statements*), others yield relational information (like *1879 births*) and again others indicate merely thematic vicinity (like *Physics*).

Identifying Conceptual Categories. Only the conceptual categories are candidates for serving as a class for the individual. The administrative and relational categories are very few (less than a dozen) and can be excluded by hand. To distinguish the conceptual categories from the thematic ones, we employ a shallow linguistic parsing of the category name (using the Noun Group Parser of [26]). For example, a category name like *Naturalized citizens of the United States* is broken into a pre-modifier (*Naturalized*), a head (*citizens*) and a post-modifier (*of the United States*). Heuristically, we found that if the head of the category name is a plural word, the category is most likely a conceptual category. We used the Pling-Stemmer from [26] to reliably identify and stem plural words. This gives us a (possibly empty) set of conceptual categories for each Wikipedia page. Conveniently, articles that do not describe individuals (like hub pages) do not have conceptual categories. Thus, the conceptual categories yield not only the TYPE relation, but also, as its domain, the set of individuals. It also yields, as its range, a set of classes.

4.1.2 The subClassOf relation

The Wikipedia categories are organized in a directed acyclic graph, which yields a hierarchy of categories. This hierarchy, however, reflects merely the thematic structure of the Wikipedia pages (e.g., as mentioned in the introduction, Zidane is in the category *Football in France*). Thus, the hierarchy is of little use from an ontological point of view. WordNet, in contrast, offers an ontologically well-defined taxonomy of synsets. Hence we use WordNet to establish the hierarchy of classes in YAGO.

Integrating WordNet Synsets. Each synset of WordNet becomes a class of YAGO. Care is taken to exclude the proper nouns known to WordNet, which in fact would be individuals (Albert Einstein, e.g., is also known to WordNet, but excluded). There are roughly 15,000 cases, in which an entity is contributed by both WordNet and Wikipedia (i.e. a WordNet synset contains a common noun that is the name of a Wikipedia page). In some of these cases, the Wikipedia page describes an individual that bears a common noun as its name (e.g. "Time exposure" is a common noun for WordNet, but an album title for Wikipedia). In the overwhelming majority of the cases, however, the Wikipedia page is simply about the common noun (e.g. the Wikipedia page "Physicist" is about physicists). To be on the safe side, we always give preference to WordNet and discard the Wikipedia individual in case of a conflict. This way, we lose information about individuals that bear a common noun as name, but it ensures that all common nouns are classes and no entity is duplicated.

Establishing subClassOf. The SUBCLASSOF hierarchy of classes is taken from the hyponymy relation from WordNet: A class is a subclass of another one, if the first synset is a hyponym of the second. Now, the lower classes extracted from Wikipedia have to be connected to the higher classes extracted from WordNet. For example, the Wikipedia class *American people* *in Japan* has to be made a subclass of the WordNet class *person.* To this end, we use the following algorithm:

```
Function wiki2wordnet($c$)
Input: Wikipedia category name $c$
Output: WordNet synset
1   $head$ =headCompound($c$)
2   $pre$ =preModifier($c$)
3   $post$ =postModifier($c$)
4   $head$ =stem($head$)
5   If there is a WordNet synset $s$ for $pre + head$
6      return $s$
7   If there are WordNet synsets $s_1, ...s_n$ for $head$
8                 (ordered by their frequency for $head$)
9      return $s_1$
10  fail
```

We first determine the head compound, the pre-modifier and the post-modifier of the category name (lines 1-3). For the Wikipedia category *American people in Japan*, these are "American", "people" and "in Japan", respectively. We stem the head compound of the category name (i.e. *people*) to its singular form (i.e. *person*) in line 4. Then we check whether there is a WordNet synset for the concatenation of pre-modifier and head compound (i.e. *American person*). If this is the case, the Wikipedia class becomes a subclass of the WordNet class (lines 5-6). If this is not the case, we exploit that the Wikipedia category names are almost exclusively endocentric compound words (i.e. the category name is a hyponym of its head compound, e.g. "American person" is a hyponym of "person"). The head compound (*person*) has to be mapped to a corresponding WordNet synset ($s_1, ..., s_n$ in line 7). This mapping is non-trivial, since one word may refer to multiple synsets in WordNet. We experimented with different disambiguation approaches. Among others, we mapped the co-occurring categories of a given category to their possible synsets as well and determined the smallest subgraph of synsets that contained one synset for each category. WordNet stores with each word the frequencies with which it refers to the possible synsets. We found out that mapping the head compound simply to the most frequent synset (s_1) yields the correct synset in the overwhelming majority of cases. This way, the Wikipedia class *American people in Japan* becomes a subclass of the WordNet class *person/human.*

Exceptions. There were only a dozen prominent exceptions, which we corrected manually. For example, all categories with the head compound *capital* in Wikipedia mean the "capital city", but the most frequent sense in WordNet is "financial asset". In summary, we obtain a complete hierarchy of classes, where the upper classes stem from WordNet and the leaves come from Wikipedia.

4.1.3 The means relation

Exploiting WordNet Synsets. Wikipedia and WordNet also yield information on word meaning. WordNet for example reveals the meaning of words by its synsets. For example, the words ”urban center” and ”metropolis” both belong to the synset *city*. We leverage this information in two ways. First, we introduce an entity for each noun known to WordNet (i.e. ”physicist”). Second, we establish a MEANS relation between each word of synset and the corresponding class (i.e. (”physicist”, MEANS, *physicist*)).

Exploiting Wikipedia Redirects. Wikipedia contributes names for the individuals by its redirect system: a Wikipedia redirect is a virtual Wikipedia page, which links to a real Wikipedia page. These links serve to redirect users to the correct Wikipedia article. For example, if the user typed ”Einstein, Albert” instead of ”Albert Einstein”, then there is a virtual redirect page for ”Einstein, Albert” that links to ”Albert Einstein”. We exploit the redirect pages to give us alternative names for the entities. For each redirect, we introduce a corresponding MEANS fact (e.g. (”Einstein, Albert”, MEANS, `Albert Einstein`)).

Parsing Person Names. The YAGO hierarchy of classes allows us to identify individuals that are persons. If the words used to refer to these individuals match the common pattern of a given name and a family name, we extract the name components and establish the relations GIVENNAMEOF and FAMILYNAMEOF. For example, we know that `Albert Einstein` is a person, so we introduce the facts (”Einstein”, FAMILYNAMEOF, `Albert Einstein`) and (”Albert”, GIVENNAMEOF, `Albert Einstein`). Both are subrelations of MEANS, so that the family name ”Einstein”, for example, also means `Albert Einstein`. We used the Name Parser from [26] to identify and decompose the person names.

4.1.4 Other relations

Exploiting Wikipedia Categories. We exploit relational Wikipedia categories for the extraction of the following relations: BORNINYEAR, DIEDINYEAR, ESTABLISHEDINYEAR, LOCATEDIN, WRITTENINYEAR, POLITICIANOF, and HASWONPRIZE. For the extraction of the BORNINYEAR and DIEDINYEAR facts we make use of the categories ending with “*_births*” and “*_deaths*” respectively. For example, if a page is in the category “*1879_births*”, it means that the corresponding individual is a person born in 1879.

The ESTABLISHEDINYEAR facts are extracted from categories ending with "*_establishments*". For example, if a page is in the category *1980_establishments*, this means that the corresponding individual (mostly an organization) was established in 1980. We normalize vague date expressions (like "*5'th_century_BC*") to a common form (e.g. -500).

The LOCATEDIN facts are extracted from categories that imply that all its members share a geographical location. For example, if a page is in the category *Cities_in_Germany*, this indicates that the corresponding individual is located in Germany. We make use of categories starting with *Countries in...*, *Rivers of...*, *Attractions in...* and similar ones. Note that we do not need to extract the classes for the individuals, since this information has already been extracted within the scope of the TYPE relation.

Further relations that we considered are the WRITTENINYEAR relation which holds between books and the year in which they appeared, the POLITICIANOF relation which holds between politicians and states as well as the HASWONPRIZE relation which concerns prize winners and the prizes they won. The facts for these three relations were extracted analogously to the afore mentioned facts.

Filtering the Results. Not all facts extracted this way constitute valid facts in the YAGO ontology, because their arguments may not be entities, but arbitrary Wikipedia pages. This is why a cleaning step is necessary, in which we filter out all facts with arguments that are not in the domain of the previously established TYPE relation. Despite the huge number of extracted facts, the actual extraction process took only a few hours. This is because it is not necessary to access the Wikipedia pages themselves – let alone parse them or POS-tag them. All information is derived from the category lists.

4.1.5 Meta-relations

Descriptions. Due to its generality, the YAGO ontology can store meta-relations uniformly together with usual relations. For example, we store for each individual the URL of the corresponding Wikipedia page. This will allow future applications to provide the user with detailed information on the entities. We introduce the DESCRIBES relation between the individual and its URL for this purpose.

Witnesses. YAGO is prepared to be extended by new facts. If a new fact was extracted from a particular Web page, we call this page the *witness* for the fact. We introduce the FOUNDIN relation, which holds between a fact and the URL of the witness page. We use the EXTRACTEDBY relation to identify the technique by which a fact was extracted. The information about witnesses will enable applications to use, e.g., only facts extracted by a certain technique, facts extracted from a certain source or facts of a certain date.

Context. Last, we store for each individual the individuals it is linked to in the corresponding Wikipedia page. For example, `Albert Einstein` is linked to `Relativity Theory`. For this purpose, we introduce the CONTEXT relation between individuals. This will allow applications to use related terms for disambiguation purposes. Different from a simple co-occurence table of words, the CONTEXT relation connects entities instead of words, i.e. its arguments are already disambiguated.

4.2 YAGO Storage

The YAGO model itself is independent of a particular data storage format. To produce minimal overhead, we decided to use simple text files as an internal format. We maintain a folder for each relation and each folder contains files that list the entity pairs. We store only facts that cannot be derived by the rewrite rules of YAGO (see 2.2), so that we store in fact the unique canonical base of the ontology. We plan to make the files publicly available, so that the YAGO ontology can be used freely for research purposes.

Furthermore, we provide conversion programs to convert the ontology to different output formats. First, YAGO is available as a simple XML version of the text files. Furthermore, YAGO can be converted to a database table. The table has the simple schema
`FACTS(factId,arg1,relation,arg2,confidence)`. We provide software to load YAGO into an Oracle or MySQL database. For our experiments, we used the Oracle version of YAGO. Last, we also provide an RDFS version of YAGO, as explained in Chapter 2.3.

4.3 Enriching YAGO

YAGO is designed to be extendable by new facts. An application that adds new facts to the YAGO ontology is required to obey the following protocol. Suppose that the application wishes to add the fact (x, r, y). First, it has to map x and y to existing entities in the YAGO ontology. This task is essentially a word sense disambiguation problem, in which the "words" x and y have to be disambiguated to YAGO entities. For the disambiguation, the application can make use of the extensive information that YAGO provides for the existing entities: the relations to other entities, the words used to refer to the entities, and the context of the entities, as provided by the CONTEXT relation. If x or y do not yet exist in the ontology, they have to be added as new entities. Next, r has to be mapped to a relation in the YAGO ontology.

Currently, YAGO comprises only a fixed set of relations, which simplifies this task. The application should provide a confidence value $c \in [0, 1]$ for the proposed fact (x, r, y).

If (x, r, y) exists already in the ontology, the application merely adds a new witness for this fact. Supposing that the fact identifier of the existing fact is f and that the witness is w, the new fact would be $(f, \textsc{foundIn}, w)$ with confidence c. Then, the confidence of the existing fact has to be recomputed as an aggregation of the confidences of the witnesses. We propose to take the maximum, but other options can be considered. If (x, r, y) does not yet exist in the ontology, the application has to add the fact together with a new fact identifier. We propose to use name spaces as in OWL/RDFS: each application has a universally unique id and the fact identifier is composed of the application id and a running number. Chapter 5.3 shows how the enrichment can be implemented in practice.

5 Evaluation and Experiments

5.1 Manual evaluation

5.1.1 Accuracy

We were interested in the accuracy of YAGO. To evaluate the accuracy of an ontology, its facts have to be compared to some ground truth. Since there is no computer-processable ground truth of suitable extent, we had to rely on manual evaluation. We presented randomly selected facts of the ontology to anonymous human judges and asked them to assess whether the facts were correct. Since common sense often does not suffice to judge the correctness of YAGO facts, we also presented them a snippet of the corresponding Wikipedia page. Thus, our evaluation compared YAGO against the ground truth of Wikipedia (i.e., it does not deal with the problem of Wikipedia containing false information). Of course, it would be pointless to evaluate the portion of YAGO that stems from WordNet, because we can assume human accuracy here. Likewise, it would be pointless to evaluate the non-heuristic relations in YAGO, such as DESCRIBES, MEANS, or CONTEXT. This is why we evaluated only those facts that constitute potentially weak points in the ontology. To be sure that our findings are significant, we computed the Wilson intervals for $\alpha = 5\%$.

The evaluation shows very good results. Especially the crucial TYPE relation and the link between WordNet and Wikipedia, SUBCLASSOF, turned out to be very accurate. Our heuristic algorithms cannot always achieve an accuracy of 100%, tough. This may also have to do with the inconsistency of the underlying sources. For example, for the relation BORNINYEAR, most false facts stem from erroneous Wikipedia categories (e.g. some person born in 1802 is in the Wikipedia category *1805 births*). In addition, the evaluation of an ontology is sometimes a philosophical issue. To start with, even simple relations suffer from vagueness (e.g. is Lake Victoria LOCATEDIN Tanzania, if Tanzania borders the lake? Is an economist who works in France a *French Economist*, even if he was born in Ireland?). Next, it is not always clear

Table 5.1: Accuracy of YAGO

Relation	# evaluated facts	Accuracy	
subClassOf	298	97.70%	± 1.59%
type	343	94.54%	± 2.36%
familyNameOf	221	97.81%	± 1.75%
givenNameOf	161	97.62%	± 2.08%
establishedInYear	170	90.84%	± 4.28%
bornInYear	170	93.14%	± 3.71%
diedInYear	147	98.72%	± 1.30%
locatedIn	180	98.41%	± 1.52%
politicianOf	176	92.43%	± 3.93%
writtenInYear	172	94.35%	± 3.33%
hasWonPrize	122	98.47%	± 1.57%

whether an entity should be an individual or a class (e.g. a judge might decide that `physics` is an individual, because it is an instance of `science`). YAGO, however, in accordance with WordNet, sees abstract notions in general as classes, because they can have subclasses (e.g., `physics` can have the subclass `astrophysics`). Furthermore, not everybody may agree on the definition of synsets in WordNet (e.g., a `palace` is in the same synset as a `castle` in WordNet). Thus, the above results measure not only the correctness of the facts in YAGO, but also the accuracy of Wikipedia category assignments, the judges' philosophical agreement with the YAGO semantics and the judges' agreement with the definition of WordNet synsets. This type of disputability is inherent even to human-made ontologies. Thus, we can be extremely satisfied with our results. Further note that these values measure just the potentially weakest point of YAGO, as all other facts were derived non-heuristically.

It is difficult to compare YAGO to other information extraction approaches, because the approaches usually differ in the choice of relations and in the choice of the sources. YAGO is tailored to Wikipedia and WordNet, but it comes with a multitude of interconnected relations. Furthermore, accuracy can usually be varied at the cost of recall. Approaches that use pattern matching (e.g. the Espresso System [19] or Leila [25]) typically achieve accuracy rates of 50%-92%, depending on the extracted relation. State-of-the-art taxonomy induction as described in [22] achieves an accuracy of 84%. KnowItAll [9] and KnowItNow [4] are reported to have accuracy rates of 85% and 80%, respectively.

5.1.2 Coverage

Table 5.2 shows the number of facts for each relation in YAGO. The overall number of ontological facts is about 5 million. This number is completed by the respective witness facts and approximately 40 million context facts.

Table 5.2: Coverage of YAGO (facts)

Relation	**Domain**	**Range**	**# Facts**
subClassOf	class	class	126,792
type	entity	class	2,011,072
context	entity	entity	~40,000,000
describes	word	entity	997,061
bornInYear	person	year	189,950
diedInYear	person	year	93,827
establishedInYear	entity	year	14,602
locatedIn	object	region	60,354
writtenInYear	book	year	4,399
politiciansOf	organization	person	3,618
hasWonPrize	person	prize	1,024
means	word	entity	2,166,891
familyNameOf	word	person	181,926
givenNameOf	word	person	177,291

Table 5.3 shows the number of entities in YAGO.

Table 5.3: Coverage of YAGO (entities)

Relations	14
Classes	84,513
Individuals (without words)	818,248

It is not easy to compare the coverage of YAGO to other ontologies, because the ontologies usually differ in their structure, their relations and their domain. For informational purposes, we list the number of entities and facts that are reported for some of the most important other domain-independent ontologies in Table 5.4.
With the exception of Cyc (which is not publicly available), the facts of these ontologies are in the hundreds of thousands, whereas the facts of YAGO are in the millions.

Table 5.4: Coverage of other ontologies

Ontology	Entities	Facts
KnowItNow [4]	N/A	25,860
KnowItAll [9]	N/A	29,835
SUMO [17]	20,000	60,000
WordNet [10]	117,597	207,016
OpenCyc [16]	47,000	306,000
Cyc [16]	250,000	2,200,000

5.2 Sample facts

Table 5.5 shows some sample facts of YAGO. In YAGO, the word "Paris", can refer to 71 distinct entities. We list some interesting ones. The football player Zinedine Zidane, e.g., is an instance of 24 different classes in our ontology. We list some of them. In the table, TYPE+SUBCLASS means that the individual is an instance of a class that is a subclass of the given class.

Table 5.5: Sample facts of YAGO

Zidane	TYPE+SUBCLASS	football player
Zidane	TYPE	Person from Marseille
Zidane	TYPE	Legion d'honneur recipient
Zidane	BORNINYEAR	1972
"Paris"	FAMILYNAMEOF	Priscilla Paris
"Paris"	GIVENNAMEOF	Paris Hilton
"Paris"	MEANS	Paris, France
"Paris"	MEANS	Paris, Texas
Paris, France	LOCATEDIN	France
Paris, France	TYPE+SUBCLASS	capital
Paris, France	TYPE	Eurovision host city
Paris, France	ESTABLISHEDINYEAR	-300

We also provide an interface to query YAGO in a SPARQL-like fashion [28] [1]. A query is a list of facts containing variables and regular expressions. Preprocessing ensures that words in the query are considered in all their possible meanings. The query algorithms are not in the scope of this paper (see [24] for details). Here, we only show some sample queries to illustrate the applicability of YAGO (Table 5.6).

[1] Available at http://www.mpii.mpg.de/∼suchanek

Table 5.6: Sample queries on YAGO

Query	**Result**
When was "Mostly Harmless" written? (`Mostly_Harmless`,WRITTENINYEAR,`$y`)	\$y=1992
Which humanists were born in 1879? (`$h`, TYPE SUBCLASSOF*, `humanist`) (`$h`, BORNINYEAR, `1879`)	\$h=Albert_Einstein and 2 more
Which locations in Texas and Illinois bear the same name? (`$t`, LOCATEDIN, `Texas`) (`$n`, MEANS, `$t`) (`$n`, MEANS, `$k`) (`$k`, LOCATEDIN, `Illinois`)	\$n="Farmersville" and 121 more

5.3 Enrichment experiment

To demonstrate how an application can add new facts to the YAGO ontology, we conducted an experiment with the knowledge extraction system LEILA [25]. LEILA is a state-of-the-art system that uses pattern matching on natural language text. It can extract facts of a certain given relation from Web documents. We trained LEILA for the HEADQUARTEREDIN relation (as described in [26]). This relation holds between a company and the city of its headquarters. We ran LEILA on a corpus of 150 news documents and had it extract pairs of companies and headquarters (see [26] for details). Each extracted pair is a candidate fact (e.g. if LEILA extracted the pair `Microsoft` / `Redmond`, then (`Microsoft`, HEADQUARTEREDIN, `Redmond`) is a candidate fact). Since the HEADQUARTEREDIN relation is not part of YAGO, no candidate fact is already present in YAGO. For each candidate fact, the company and the city have to be mapped to the respective individuals in YAGO.

To disambiguate the company name, we proceeded as follows: By the MEANS relation, one can find out which individuals in YAGO the company name refers to. If exactly one of these individuals is an instance of the class `company`, we map the company name to this individual. If multiple individuals are instances of the class `company`, we cannot be sure which one is meant. In this case, we abandon the candidate fact, because we aim at a high accuracy at the cost of a potentially lower coverage (this did not happen in our experiment). If no individual is a `company`, we introduce the company name as a new individual for YAGO.

To disambiguate the city name, we proceed similarly. We identify a set of potential individuals by the MEANS relation together with the constraint that the individual be a `city`. If no individual is a `city`, we abandon the fact because we assume that Wikipedia knows all major cities. If multiple individuals are a `city`, we use a simple disambiguation heuristic: We pick the city located in the state that is mentioned most frequently in the article. If no such city exists, the fact is abandoned. This way, both the company and the city get mapped to the respective YAGO individuals.

The confidence of the new fact is computed as a normalization of the confidence score returned by LEILA. Table 5.7 shows the number of company names and HEADQUARTEREDIN facts contributed by LEILA.

Table 5.7: LEILA HEADQUARTEREDIN facts

Abandoned candidates	
because of an unknown city	32
because of an ambiguous city	33
because of an ambiguous company	0
Total	65
Inserted candidates	
with a known company name	10
with a new company name	70
Total	80

In the above merging process, entities that are already present in the ontology help to disambiguate new entities. Thus, the more facts and entities YAGO contains, the better it can be extended by new facts. The better YAGO can be extended, the more facts it will contain. This mutual contribution constitutes a positive re-enforcement loop, which could help future expansion.

6 Conclusion

In this paper, we presented YAGO, a light-weight and extendable ontology of high quality and coverage. YAGO contains 900,000 entities and 5 million facts – more than any other publicly available formal ontology. As our evaluation shows, YAGO has a near-human accuracy around 95%.

Our data model defines a clear semantics for YAGO. It is decidable and it guarantees that the smallest ontology in a set of equivalent ontologies is unique, so that there is a canonical way to store a YAGO ontology.

We demonstrated how YAGO can be extended by facts extracted from Web documents through state-of-the-art extraction techniques. We observed that the more facts YAGO contains, the easier it is to extend it by further facts. This positive feedback loop could facilitate the growth of the knowledge base. YAGO will be made available in different export formats, including plain text, XML, RDFS and SQL database formats.

YAGO opens the door to numerous new challenges. On the theoretical side, we plan to investigate the relationship between OWL 1.1 and the YAGO model, once OWL 1.1 has been fully developed. This might necessitate extensions or additional semantic restrictions of the YAGO model. On the practical side, we plan to enrich YAGO by further facts – including high confidence facts from gazetteers, but also extracted information from Web pages. In particular, we envisage to analyze and exploit the positive feedback loop of data gathering. We hope that the availability of a huge, clean, and high quality ontology can give new impulses to the Semantic Web vision.

Appendix A Proof of Theorem 1

Be $\mathcal{F}$ a (finite) set of fact triples, as defined in Chapter 2.2. Be $\rightarrow$ the rewrite system defined there (see [2] for a reference on term rewriting). All rules of the rewrite system are of the form $F \rightarrow F \cup \{f\}$, where $F \subseteq \mathcal{F}$ and $f \in \mathcal{F}$. Hence $\rightarrow$ is monotone. Furthermore, $\mathcal{F}$ is finite. Hence $\rightarrow$ is finitely terminating. It is easy to see that if $F \rightarrow F \cup \{f_1\}$ and $F \rightarrow F \cup \{f_2\}$ for some $F \subseteq \mathcal{F}$ and $f_1, f_2 \in \mathcal{F}$, then

$$\begin{aligned} &F \rightarrow F \cup \{f_1\} \rightarrow F \cup \{f_1, f_2\} \\ &F \rightarrow F \cup \{f_2\} \rightarrow F \cup \{f_1, f_2\} \end{aligned}$$

Hence $\rightarrow$ is locally confluent. Since $\rightarrow$ is finitely terminating, $\rightarrow$ is globally confluent and convergent. Thus, given any set of facts $F \subseteq \mathcal{F}$, the largest set D_F with $F \rightarrow^* D_F$ is unique and finite.

Appendix B Proof of Theorem 2

A *canonical base* of a YAGO ontology y is any base b of y, such that there exists no other base b' of y with $|b'| < |b|$. This section will prove that, for a consistent YAGO ontology, there exists exactly one such base. In the following, $\rightarrow$ denotes the rewrite system and $\mathcal{F}$ denotes the set of facts defined in Chapter 2.2.

LEMMA 1: [*No circular rules*]
Be y a consistent YAGO ontology, be $\{f_1, ..., f_n\}$ a set of facts. Then there are no sets of facts $F_1, ..., F_n$, such that that $F_1, ..., F_n \subseteq D(y)$ and

$$\begin{array}{lll} F_1 \hookrightarrow f_1 & \text{with} & f_2 \in F_1 \\ F_2 \hookrightarrow f_2 & \text{with} & f_3 \in F_2 \\ ... & & \\ F_n \hookrightarrow f_n & \text{with} & f_1 \in F_n \end{array}$$

Proof: By analyzing all possible pairs of rule schemes (1)...(5), one finds that the above rules must fall into one of the following categories:

- All rules are instances of (5). In this case, $(c, \text{SUBCLASSOF}, c) \in D(y)$ for some common entity c and hence y cannot be consistent.
- All rules are instances of (1). In this case, $(c, \text{SUBRELATIONOF}, c) \in D(y)$ for some common entity c and hence y cannot be consistent.
- All rules are instances of (2). In this case, $(c, r, c) \in D(y)$ for some common entity c and relation r and (r,TYPE,`acyclicTransitive-Relation`)$\in D(y)$ and hence y cannot be consistent.
- $n = 2$, one rule is an instance of (1), and the other an instance of (2). In this case, $(c, r, c) \in D(y)$ for some common entity c and relation r and

(r,TYPE,`acyclicTransitiveRelation`)$\in D(y)$ and hence y cannot be consistent.

LEMMA 2: [*No derivable facts in canonical base*]
Be y a consistent YAGO ontology. Be b a canonical base of y and $B = range(b)$. Be $f \in D(y)$ a fact such that $D(y)\backslash\{f\} \rightarrow D(y)$. Then $f \notin B$.

Proof: Since b is a base, there is a sequence of sets of facts $B_0, ..., B_n$ such that

$$B = B_0 \rightarrow B_1 \rightarrow B_2 \rightarrow \ldots \rightarrow B_{n-1} \rightarrow B_n = D(y)$$

This sequence is a sequence of rule applications, where each rule has the form $S \hookrightarrow s$, where $S \subseteq \mathcal{F}$ and $s \in \mathcal{F}$. We call S the *premise* of the rule and s its *conclusion*. We say that a fact t *contributes* to a set of facts T in the sequence $B_0, ...B_n$, if there is a sequence of rule applications $r_1, ...r_m$, so that t is in the premise of r_1, the conclusion of r_1 is in the premise of r_2 etc. and the conclusion of r_m is in T.

Now assume $f \in B$. Since $D(y)\backslash\{f\} \rightarrow D(y)$, there must be a rule $G \hookrightarrow f$ with $G \subseteq D(y)\backslash\{f\}$. Be $i \in [0, n]$ the smallest index such that $B_i \supseteq G$. f cannot contribute to G, because then there would exist circular rules in the sense of the preceding lemma. Hence f does not contribute to G. Then $B\backslash\{f\}$ is also a base, because the above rule applications can be re-ordered so that f is derived from B_i. Hence b cannot be a canonical base.

Now we are ready to prove Theorem 2:

THEOREM 2: [*Uniqueness of the Canonical Base*]
The canonical base of a consistent YAGO ontology is unique.

Proof: Be b a canonical base of a consistent YAGO ontology y. Be $B = range(b)$. We define the set

$$C := D(y) \setminus \{f \mid D(y)\backslash\{f\} \rightarrow D(y)\}$$

Intuitively speaking, C contains only those facts that cannot be derived from other facts in $D(y)$. By the previous lemma, $B \subseteq C$. Assume $B \subset C$, i.e. there exists a fact $f \in C$, $f \notin B$. Since $C \subseteq D(y)$, $f \in D(y)$. Since b is a base, there exists a rule $S \hookrightarrow f$ for some $S \subseteq D(y)$. Hence $f \notin C$, which is a contradiction. Hence $B = C$ and every canonical base equals b.

This theorem entails that the canonical base of a YAGO ontology can be computed by removing all facts that can be derived from other facts in the set of derivable facts.

Bibliography

[1] E. Agichtein and L. Gravano. *Snowball*: extracting relations from large plain-text collections. In *ICDL*, 2000.

[2] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998.

[3] R. C. Bunescu and M. Pasca. Using encyclopedic knowledge for named entity disambiguation. In *EACL*, 2006.

[4] M. J. Cafarella, D. Downey, S. Soderland, and O. Etzioni. KnowItNow: Fast, scalable information extraction from the web. In *EMNLP*, 2005.

[5] N. Chatterjee, S. Goyal, and A. Naithani. Resolving pattern ambiguity for english to hindi machine translation using WordNet. In *Workshop on Modern Approaches in Translation Technologies*, 2005.

[6] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *ICDE*, 2005.

[7] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *KDD*, 2004.

[8] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *ACL*, 2002.

[9] O. Etzioni, M. J. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll. In *WWW*, 2004.

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

[11] J. Graupmann, R. Schenkel, and G. Weikum. The spheresearch engine for unified ranked retrieval of heterogeneous XML and web documents. In *VLDB*, 2005.

[12] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *KR*, 2006.

[13] W. Hunt, L. Lita, and E. Nyberg. Gazetteers, wordnet, encyclopedias, and the web: Analyzing question answering resources. Technical Report CMU-LTI-04-188, Language Technologies Institute, Carnegie Mellon, 2004.

[14] G. Ifrim and G. Weikum. Transductive learning for text classification using explicit knowledge models. In *PKDD*, 2006.

[15] S. Liu, F. Liu, C. Yu, and W. Meng. An effective approach to document retrieval via utilizing wordnet and recognizing phrases. In *SIGIR*, 2004.

[16] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira. An introduction to the syntax and content of Cyc. In *AAAI Spring Symposium*, 2006.

[17] I. Niles and A. Pease. Towards a standard upper ontology. In *FOIS*, 2001.

[18] N. F. Noy, A. Doan, and A. Y. Halevy. Semantic integration. *AI Magazine*, 26(1):7–10, 2005.

[19] P. Pantel and M. Pennacchiotti. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *ACL*, 2006.

[20] M. Ruiz-Casado, E. Alfonseca, and P. Castells. Automatic extraction of semantic relationships for WordNet by means of pattern learning from Wikipedia. In *NLDB*, pages 67–79, 2006.

[21] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice Hall, 2002.

[22] R. Snow, D. Jurafsky, and A. Y. Ng. Semantic taxonomy induction from heterogenous evidence. In *ACL*, 2006.

[23] S. Staab and R. Studer. *Handbook on Ontologies*. Springer, 2004.

[24] F. Suchanek, G. Kasneci, M. Ramanath, and G. Weikum. Naga: Uncoiling the Web. Research Report MPI-I-2006-5-007, Max-Planck-Institut für Informatik, Germany, 2006.

[25] F. M. Suchanek, G. Ifrim, and G. Weikum. Combining linguistic and statistical analysis to extract relations from web documents. In *KDD*, 2006.

[26] F. M. Suchanek, G. Ifrim, and G. Weikum. LEILA: Learning to Extract Information by Linguistic Analysis. In *Workshop on Ontology Population at ACL/COLING*, 2006.

[27] M. Theobald, R. Schenkel, and G. Weikum. TopX and XXL at INEX 2005. In *INEX*, 2005.

[28] W3C. Sparql, 2005. retrieved from http://www.w3.org/TR/rdf-sparql-query/.

MPI-I-2006-5-006	F.M. Suchanek, G. Kasneci, G. Weikum	Yago: A Core of Semantic Knowledge
MPI-I-2006-RG1-001	S. Hirth, C. Karl, C. Weidenbach	Automatic Infrastructure for ..... Analysis
MPI-I-2006-5-005	R. Angelova, S. Siersdorfer	A Neighborhood-Based Approach for Clustering of Linked Document Collections
MPI-I-2006-5-004	F. Suchanek, G. Ifrim, G. Weikum	Combining Linguistic and Statistical Analysis to Extract Relations from Web Documents
MPI-I-2006-5-003	V. Scholz, M. Magnor	Garment Texture Editing in Monocular Video Sequences based on Color-Coded Printing Patterns
MPI-I-2006-5-002	H. Bast, D. Majumdar, R. Schenkel, M. Theobald, G. Weikum	IO-Top-k: Index-access Optimized Top-k Query Processing
MPI-I-2006-5-001	M. Bender, S. Michel, G. Weikum, P. Triantafilou	Overlap-Aware Global df Estimation in Distributed Information Retrieval Systems
MPI-I-2006-4-010	A. Belyaev, T. Langer, H. Seidel	Mean Value Coordinates for Arbitrary Spherical Polygons and Polyhedra in R^3
MPI-I-2006-4-009	J. Gall, J. Potthoff, B. Rosenhahn , C. Schnoerr, H. Seidel	Interacting and Annealing Particle Filters: Mathematics and a Recipe for Applications
MPI-I-2006-4-008	I. Albrecht, M. Kipp, M. Neff, H. Seidel	Gesture Modeling and Animation by Imitation
MPI-I-2006-4-007	O. Schall, A. Belyaev, H. Seidel	Feature-preserving Non-local Denoising of Static and Time-varying Range Data
MPI-I-2006-4-006	C. Theobalt, N. Ahmed, H. Lensch, M. Magnor, H. Seidel	Enhanced Dynamic Reflectometry for Relightable Free-Viewpoint Video
MPI-I-2006-4-005	A. Belyaev, H. Seidel, S. Yoshizawa	Skeleton-driven Laplacian Mesh Deformations
MPI-I-2006-4-004	V. Havran, R. Herzog, H. Seidel	On Fast Construction of Spatial Hierarchies for Ray Tracing
MPI-I-2006-4-003	E. de Aguiar, R. Zayer, C. Theobalt, M. Magnor, H. Seidel	A Framework for Natural Animation of Digitized Models
MPI-I-2006-4-002	G. Ziegler, A. Tevs, C. Theobalt, H. Seidel	GPU Point List Generation through Histogram Pyramids
MPI-I-2006-4-001	R. Mantiuk	?
MPI-I-2006-2-001	T. Wies, V. Kuncak, K. Zee, A. Podelski, M. Rinard	On Verifying Complex Properties using Symbolic Shape Analysis
MPI-I-2006-1-007	I. Weber	?
MPI-I-2006-1-006	M. Kerber	Division-Free Computation of Subresultants Using Bezout Matrices
MPI-I-2006-1-004	E. de Aguiar	?
MPI-I-2006-1-001	M. Dimitrios	?
MPI-I-2005-5-002	S. Siersdorfer, G. Weikum	Automated Retraining Methods for Document Classification and their Parameter Tuning
MPI-I-2005-4-006	C. Fuchs, M. Goesele, T. Chen, H. Seidel	An Emperical Model for Heterogeneous Translucent Objects