# Yago: A Large Ontology from Wikipedia and WordNet

Fabian M. Suchanek, Gjergji Kasneci, Gerhard Weikum

**Authors' Addresses**

Fabian M. Suchanek
Max-Planck-Institute for Computer Science
Campus E1 4
66123 Saarbrücken
Germany

Gjergji Kasneci
Max-Planck-Institute for Computer Science
Campus E1 4
66123 Saarbrücken
Germany

Gerhard Weikum
Max-Planck-Institute for Computer Science
Campus E1 4
66123 Saarbrücken
Germany

## Abstract

This article presents YAGO, a large ontology with high coverage and precision. YAGO has been automatically derived from Wikipedia and WordNet. It comprises entities and relations, and currently contains more than 1.7 million entities and 15 million facts. These include the taxonomic Is-A hierarchy as well as semantic relations between entities. The facts for YAGO have been extracted from the category system and the infoboxes of Wikipedia and have been combined with taxonomic relations from WordNet. Type checking techniques help us keep YAGO's precision at 95% – as proven by an extensive evaluation study. YAGO is based on a clean logical model with a decidable consistency. Furthermore, it allows representing n-ary relations in a natural way while maintaining compatibility with RDFS. A powerful query model facilitates access to YAGO's data.

## Keywords

# Contents

# 1 Introduction

Many applications in modern information technology utilize ontological background knowledge. This applies above all to applications in the vision of the Semantic Web, but also to numerous other application fields: machine translation (e.g. [34]) and word sense disambiguation (e.g. [26]) exploit lexical knowledge; query expansion uses taxonomies (e.g. [97, 71, 142]); document classification based on supervised or semi-supervised learning can be combined with ontologies (e.g. [80]); and [79] 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) [35], record linkage (aka. entity resolution) [46], and information integration in general [115]. In addition, there are emerging trends towards entity- and fact-oriented Web search and community management [16, 28, 33, 36, 52, 87, 99, 109, 111], which can build on rich knowledge bases.

But the existing applications typically use only a single source of background knowledge (mostly WordNet [66] 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.[1] It would have to be 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] In this article, we mean by "ontology" any set of facts and/or axioms, comprising potentially both individuals and concepts.

## 1.1 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 [124] and [131]). Numerous approaches have been proposed to create general-purpose ontologies on top of these representations. One class of approaches focuses on extracting knowledge structures automatically from text corpora. These approaches use information extraction technologies that include pattern matching, natural-language parsing, and statistical learning [137, 65, 27, 2, 128, 116, 50]. These techniques have also been used to extend WordNet by Wikipedia individuals [123]. Two important projects along these lines are KnowItAll [65] and TextRunner [13]. KnowItAll aimed at extracting and compiling instances of a given set 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. TextRunner has the even more ambitious goal of extracting all instances of *all* meaningful relations from Web pages, a paradigm referred to as *machine reading* [63]. Recently this approach has been further extended to include even *lifelong learning*, by using the already compiled knowledge to drive the strategies for acquiring new facts [15]. Although automatic knowledge acquisition of this kind often exhibits results of remarkable accuracy, the quality is still significantly below that of a hand-crafted knowledge base. Furthermore, these systems do not extract facts in a canonical form, i.e. different identifiers are used for the same entity and there exist no clearly defined relations. As a result, no explicit (logic-based) knowledge representation model is available. Thus, information-extraction approaches are still much more suitable for high coverage and less attractive for applications that need consistent ontologies (e.g., high-accuracy query processing, or even automated reasoning).

Similar observations hold for the recently popularized direction of mining taxonomies and semantic relations from social-tagging platforms such as `del.icio.us` and Web directories such as `dmoz.org` (see, e.g., [54, 82, 62]). Notwithstanding the benefits of these approaches, the inherent noise and lack of explicit quality control for social tagging usually lead to poor precision.

Because of the quality bottleneck, the most successful and widely employed ontologies are still man-made. These include WordNet [66], Cyc or OpenCyc [106], SUMO [114], and especially domain-specific ontologies and taxonomies such as UMLS[2] or the GeneOntology[3]. These knowledge sources have the advantage of satisfying the highest quality expectations, because

---

[2]`http://umlsinfo.nlm.nih.gov`
[3]`http://www.geneontology.org/`

they are manually assembled. However, they are costly to assemble and continuous human effort is needed to keep them up to date. As a result, no hand-crafted ontology knows the most recent Windows version or the latest soccer star.

Lately, a new approach has entered the scene: community-based ontology building. Inspired by Wikipedia, the Freebase project[4] aims to construct an ontology by inviting volunteers to contribute facts. However, it is not clear how one can keep such a "social-wisdom" ontology consistent, as conflicting facts seem almost inevitable in this kind of mass collaboration. One may argue that Wikipedia is successfully maintaining its high quality despite this difficulty, but on the other hand, Wikipedia by itself (i.e., without the added-value-creating methods presented in this paper) is not offering explicit facts but mostly stays at the level of natural-language text.

The Semantic Wikipedia project [145] is a comparable initiative. It invites Wikipedia authors to add semantic tags to their articles in order to turn the page link structure of Wikipedia into a huge semantic network. Again, the usefulness of this approach will depend on the acceptance of the project by the community and on finding successful ways of quality control.

Finally, a recently emerging approach is to automatically derive explicit facts from - the "automatically understandable" part of - Wikipedia. This direction includes DBpedia [9], Isolde [147], the work of Ponzetto et al. [118], KYLIN [148], and also our own YAGO project (with first results given in [139] and an extended, full account presented in this article). The DBpedia project was initially started by extracting facts from the infoboxes of particular types of Wikipedia articles (e.g., on people, cities, companies, music bands, etc.). In contrast to YAGO, DBpedia itself does not connect the facts into a coherent whole by a taxonomy. Furthermore, it does not use defined relations with ranges and domains, but the words from the infoboxes, so that the same relationship appears with different names (e.g. `length`, `length-in-km`, `length-km`). Thus, the consistency and accuracy of DBpedia are unknown. The aim of DBpedia is rather to interlink existing ontologies into a huge knowledge portal. YAGO has already been fed into the project.

Ponzetto et al. use rich heuristics to derive a taxonomy from Wikipedia categories and links between them. Isolde extracts class candidates from a specific domain corpus. It exploits Web sources such as Wikipedia and Wiktionary to derive additional knowledge about these candidates. Although both of these approaches pursue similar goals as YAGO, they lead to lower quality and more restricted scope.

Finally, KYLIN starts out with extraction techniques on infoboxes, similar to those of DBpedia, but then uses powerful learning techniques to auto-

---

[4]`http://www.freebase.com`

matically fill in missing values in incomplete infoboxes. The accuracy of the extraction is remarkable. Its goal, however, is filling infoboxes rather than constructing an ontological knowledge base.

## 1.2   Contributions and Outline

We present the ontology YAGO[5], which combines high coverage with high quality. Its core is assembled from one of the most comprehensive lexicons available today, Wikipedia. But rather than using natural language processing on the articles of Wikipedia, our approach builds on Wikipedia's *infoboxes* and *category pages*. Infoboxes are standardized tables that contain basic information about the entity described in the article. For example, there are infoboxes for countries, which contain the native name of the country, its capital and its size. As shown in [9], infoboxes are much easier to parse and exploit than natural language text. Category pages are lists of articles that belong to a specific category (e.g., Elvis is in the category of American rock singers). These lists give us candidates for entities (e.g. Elvis) candidates for concepts (e.g. IsA(Elvis, rockSinger)) [91] and candidates for relations (e.g. nationality(Elvis, American)). 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, Elvis is in the super-category named *Grammy Awards*, but Elvis is a Grammy Award *winner* and not a Grammy Award. WordNet, in contrast, provides a clean and carefully assembled hierarchy of thousands of concepts. But the Wikipedia concepts have no obvious counterparts in WordNet.

We present techniques that link the two sources with high 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 a precision of 95%. 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 1.7 million entities and 15 million facts about them.

We explain how we can enforce the high accuracy of our extraction heuristics through *type checking*. Type checking leverages the information that has already been extracted to verify the plausibility of newly extracted data. We show that type checking can be used both in a *reductive fashion* (eliminating facts that are implausible) and in an *inductive fashion* (adding supplemental facts so that the ontology becomes consistent). We have conducted an extensive evaluation study, which proves that YAGO has an overall precision

---

[5]Yet Another Great Ontology

6

of 95%.

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 facts (e.g., which fact was found on which Web site), n-ary relations (e.g. that Elvis won the Grammy Award in 1967) and general properties of relations (e.g., transitivity or acyclicity). We show that, despite its expressiveness, the YAGO data model is still decidable and can be mapped to RDFS. Furthermore, we present a query language as a natural extension of our data model, which allows querying reified facts.

YAGO was first presented in [139]. This article significally extends the previous work by adding the exploitation of infoboxes, introducing quality control techniques and defining a new query language. The rest of this article is organized as follows. In Section 2 we introduce YAGO's data model. Section 3 describes the sources from which the current YAGO is assembled, namely Wikipedia and WordNet. In Section 4 we explain the information extraction algorithms behind YAGO. Section 5 presents an evaluation, a comparison to other ontologies, and sample facts from YAGO. We conclude with a summary and an outlook in Section 6.

# 2 The YAGO Model

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 [131]. Its most expressive variant, OWL-full, can express properties of relations, but is undecidable. The weaker variants of OWL, OWL-lite and OWL-DL, cannot express relations between facts. RDFS, the basis of OWL, can express relations between facts, but provides only very primitive semantics. For example, it does not know transitivity, which is crucial for partial orders such as SUBCLASSOF or LOCATEDIN. 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 simple and decidable. We will first describe the YAGO model informally and then give a formal definition.

## 2.1 Informal description

As in OWL and RDFS, all objects (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 Elvis won a Grammy Award, we say that the entity `Elvis Presley` stands in the HASWONPRIZE relation with the entity `Grammy Award`. We write

    Elvis Presley  HASWONPRIZE  Grammy Award

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 Elvis was born in 1935, we write:

    Elvis Presley  BORNINYEAR  1935

Entities are abstract ontological objects, 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:

$$\text{"}\textit{Elvis}\text{"}\quad \textsc{means}\quad \texttt{Elvis Presley}$$

This allows us to deal with synonymy and ambiguity. The following line says that "Elvis" may also refer to the English songwriter Elvis Costello:

$$\text{"}\textit{Elvis}\text{"}\quad \textsc{means}\quad \texttt{Elvis Costello}$$

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

$$\texttt{Elvis Presley}\quad \textsc{type}\quad \texttt{singer}$$

Classes are also entities. Thus, each class is itself an instance of a class, namely of the class `class`[1]. Classes are arranged in a taxonomic hierarchy, expressed by the SUBCLASSOF relation:

$$\texttt{singer}\quad \textsc{subClassOf}\quad \texttt{person}$$

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 an acyclic transitive relation (`atr`):

$$\texttt{subclassOf}\quad \textsc{type}\quad \texttt{atr}$$

Acyclic transitive relations are of particular importance to YAGO because they are used to model partial orders such as SUBCLASSOF and LOCATEDIN. 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 derived from a certain source. For example, suppose that the above fact (`Elvis Presley`, BORNINYEAR, `1935`) had the fact identifier `#1`, then the following line would say that this fact was found in Wikipedia:

$$\texttt{#1}\quad \textsc{foundIn}\quad \texttt{Wikipedia}$$

---

[1]For the YAGO model, classes should be thought of as abstract identifiers rather than sets.

We will refer to entities that are neither facts nor relations as *common entities*. Common entities that are not classes will be called *individuals*. Summing up, a YAGO ontology is basically a function that maps fact identifiers to fact triples. More formally, a YAGO ontology can be described as a *reification graph*.

## 2.2 Reification graphs

A *reification graph* is defined over

- a set of nodes $N$. In YAGO, these are the common entities.

- a set of edge identifiers $I$. In YAGO, these are the fact identifiers.

- a set of labels $L$. In YAGO, these are the relation names.

The reification graph is an injective total function

$$G_{N,I,L} : I \rightarrow (N \cup I) \times L \times (N \cup I).$$

We call the range of this function the *edges* of the graph. Intuitively speaking, the edges of a reification graph cannot only connect two nodes, but also a node and an edge or even two edges. Each edge is unique and has an identifier from $I$. Furthermore, each edge has a label from $L$. Note that a reification graph of the form $G_{N,I,L} : I \rightarrow N \times L \times N$ defines a usual directed multi-graph with nodes $N$ and labeled edges $range(G_{N,I,L})$.

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 reification graph over the set of nodes $\mathcal{I} \cup \mathcal{C} \cup \mathcal{R}$ and the set of labels $\mathcal{R}$, i.e. an injective total function

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

We write down a YAGO ontology (and in general any reification graph) by listing the elements of the function in the form

$$
\begin{array}{ll}
id_1: & arg1_1 \ rel_1 \ arg2_1 \\
id_2: & arg1_2 \ rel_2 \ arg2_2 \\
... &
\end{array}
$$

To simplify, we will omit the part before the colon if it occurs nowhere else, assuming it to be an arbitrary fresh identifier. Furthermore, we allow the following shorthand notation

$$id_2 : \quad (arg1_1 \quad rel_1 \quad arg2_1) \quad rel_2 \quad arg2_2$$

to mean

$$
\begin{aligned}
&id_1: \quad arg1_1 \ rel_1 \ arg2_1 \\
&id_2: \quad id_1 \ rel_2 \ arg2_2
\end{aligned}
$$

where $id_1$ is a fresh identifier. Assuming left-associativity, the notation can be further simplified to

$$id_2: \quad arg1_1 \quad rel_1 \quad arg2_1 \quad rel_2 \quad arg2_2$$

For example, to state that Elvis' birth date was found in Wikipedia, we can simply write this fragment of the reification graph as

> Elvis BORNINYEAR 1935 FOUNDIN Wikipedia

## 2.3  $n$-ary Relations

Some facts require more than two arguments (for example the fact that Elvis got the Grammy Award in 1967). One common way to deal with this issue is to use $n$-ary relations (as for example in `wonPrizeInYear(Elvis, GrammyAward, 1967)`). RDFS and OWL do not allow $n$-ary relations. Therefore, the standard way to deal with this problem in these formalisms is to introduce a new binary relation for each argument (e.g. WINNER, PRIZE, YEAR). Then, an $n$-ary fact can be represented by a new *event entity* (say, `elvisGetsGrammy`) that is linked by these binary relations to all of its arguments:

| | | |
|---|---|---|
| GrammyAward | PRIZE | elvisGetsGrammy |
| Elvis | WINNER | elvisGetsGrammy |
| 1921 | YEAR | elvisGetsGrammy |

The YAGO model offers a more natural 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 WONPRIZEINYEAR-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: Elvis HASWONPRIZE Grammy Award

11

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

$$\#2: \quad \#1 \quad \text{INYEAR} \quad 1967$$

With our simplified syntax, this can as well be written as

$$\text{Elvis} \quad \text{HASWONPRIZE} \quad \text{Grammy Award} \quad \text{INYEAR} \quad 1967$$

## 2.4 Semantics

This section gives 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` and `atr` (for acyclic transitive relation). Furthermore, it must contain classes for all literals as given in Figure 1.

Figure 1: The YAGO literal classes
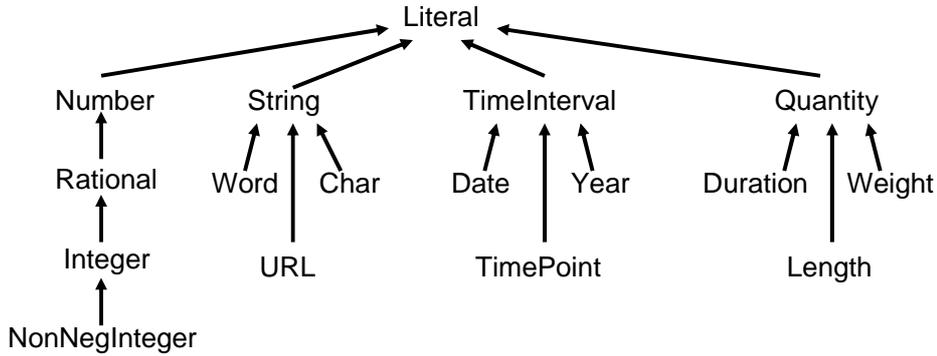


For the rest of the article, we assume a given set of common entities $\mathcal{C}$ and a given set of relations $\mathcal{R}$. The set of fact identifiers used by a YAGO ontology $y$ is implicitly given by $\mathcal{I} = domain(y)$. To define the semantics of a YAGO ontology, we consider the set of all 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:

$$\emptyset \hookrightarrow (\text{domain}, \text{RANGE}, \text{class})$$
$$\emptyset \hookrightarrow (\text{domain}, \text{DOMAIN}, \text{relation})$$
$$\emptyset \hookrightarrow (\text{range}, \text{DOMAIN}, \text{relation})$$
$$\emptyset \hookrightarrow (\text{range}, \text{RANGE}, \text{class})$$
$$\emptyset \hookrightarrow (\text{subClassOf}, \text{TYPE}, \text{atr})$$
$$\emptyset \hookrightarrow (\text{subClassOf}, \text{DOMAIN}, \text{class})$$
$$\emptyset \hookrightarrow (\text{subClassOf}, \text{RANGE}, \text{class})$$
$$\emptyset \hookrightarrow (\text{type}, \text{RANGE}, \text{class})$$
$$\emptyset \hookrightarrow (\text{subRelationOf}, \text{TYPE}, \text{atr})$$
$$\emptyset \hookrightarrow (\text{subRelationOf}, \text{DOMAIN}, \text{relation})$$
$$\emptyset \hookrightarrow (\text{subRelationOf}, \text{RANGE}, \text{relation})$$

The first rule, e.g., says that the range of the relation DOMAIN is the class `class`, i.e. the second argument of a DOMAIN fact will always be a class. In addition, the rewrite system contains for the literal classes the rules

$$\emptyset \hookrightarrow (X, \text{SUBCLASSOF}, Y)$$

for each edge $X \rightarrow Y$ in Figure 1.

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

(1) $\{(r_1, \text{SUBRELATIONOF}, r_2), (x, r_1, y)\} \hookrightarrow (x, r_2, y)$

(2) $\{(r, \text{TYPE}, \text{atr}), (x, r, y), (y, r, z)\}$
    $\hookrightarrow (x, r, z)$

(3) $\{(r, \text{DOMAIN}, c), (x, r, y)\} \hookrightarrow (x, \text{TYPE}, c)$

(4) $\{(r, \text{RANGE}, c), (x, r, y)\} \hookrightarrow (y, \text{TYPE}, c)$

(5) $\{(x, \text{TYPE}, c_1), (c_1, \text{SUBCLASSOF}, c_2)\}$
    $\hookrightarrow (x, \text{TYPE}, c_2)$

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

The proof of Theorem 1 is given in the Appendix A. 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 by a bijective substitution so that

$$(y_1 \subseteq y_2 \ \lor \ y_2 \subseteq y_1) \ \land \ D(y_1) = D(y_2)$$

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

$$y^* := y \cup \{ (f_{a,r,b}, (a, r, b)) \ | \ (a, r, b) \in D(y) \setminus range(y) \}$$

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*)

As 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* $\Psi$ with respect to a structure $< \mathcal{U}, \mathcal{D}, \mathcal{E} >$ as the following relation:

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

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 in the structure

2. if $\Psi(x, \text{TYPE}, \texttt{string})$ for some $x$, then $\mathcal{D}(x) = x$

3. if $\Psi(r, \text{TYPE}, \texttt{atr})$ 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

$$\nexists x, r : \ (r, \text{TYPE}, \texttt{atr}) \in D(y)$$
$$\wedge \ (x, r, x) \in D(y)$$

Since, by Theorem 1, the deductive closure of a YAGO ontology can be computed by applying the rules (1)-(5) finitely often, we have the following corollary of Theorem 1:

COROLLARY 1: [*Decidability*]
*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.*

The proof of Theorem 2 is given in the Appendix B. In fact, the canonical base of a YAGO ontology can be computed by greedily removing derivable facts from the ontology in any order. This makes the canonical base a natural choice to efficiently store a YAGO ontology.

## 2.5   Reification and Semantics

The YAGO model allows making statements about facts. However, it does not allow curtailing the validity of facts: A model for the ontology must make every fact true, regardless of whether the fact is an argument of another fact. This has several consequences. First, it is not possible to state in YAGO that a certain fact is false. In any case, YAGO does not provide the predefined vocabulary for such a statement and it would entail immediate undecidability. Second, the primary pair of an $n$-ary relation will always be true in a model of the ontology. Consider, for example, the fact that Elvis was a singer from 1950 to 1977. In the YAGO model, this fact could be expressed as

```
#1:  Elvis   TYPE    singer
#2:  #1     DURING   1950-1977
```

If the TYPE relation denotes the relation "$x$ is a $y$", then each model will contain the fact that Elvis is a singer – even though in the intended interpretation that holds only from 1950 to 1977. Thus, a more adequate denotation for the TYPE relation would actually be "$x$ is or was a $y$". Another consequence of the YAGO model is that intentional predicates like BELIEVESTHAT or SAYSTHAT are not possible, because all arguments to these relations would become true in the model. It does, however, allow using success verbs such as SEESTHAT or KNOWSTHAT, the arguments of which are true by intention.

These properties of the YAGO model may be considered limiting, but they guarantee the decidability of the model.

## 2.6  Data Types

The YAGO model treats literals (such as strings or numbers) as proper entities. Literals are instances of *literal classes* (or *data types*). RDFS and OWL use the data types defined by XML Schema[20]. These data types, however, are more machine-oriented and not always semantically plausible. For example, XML Schema does not know the data type `rationalNumber`, but only the disjunct data types `float` and `double`. This is why the YAGO model comes with its own data types (see Figure 1), which follow the SUMO ontology [114]. YAGO sees, e.g, `integer` as a subclass of `rational`, because each integer number is a rational number. Besides numbers, YAGO also knows `string`s. These are characterized by mapping to themselves in any denotation. `timeInterval`s are specific periods of time, such as the year 2007 or the 8th of January 1935.

The class `quantity` contains values that have a physical dimension such as length or weight. These values have units, such as meter or kilogram. In RDFS, quantities are usually represented by an anonymous entity (a *blank node*). This entity is connected by an RDF:VALUE edge to the numerical value and by a UNIT edge to the unit of measurement, e.g. as follows:

```
_:x   RDF:VALUE   1000
_:x   UNIT        gram
```

As a consequence, the very same quantity has to be represented as two blank nodes, if measured with two different units. The YAGO model, in contrast,

can express that the very same quantity has two different values if measured in different units:

| #1: | 1000g | HASVALUE | 1000 |
| #2: | #1 | INUNIT | "gram" |
| #3: | 1000g | HASVALUE | 1 |
| #4: | #3 | INUNIT | "kilogram" |

In YAGO, we use the ISO units and formats both for the HASVALUE facts and as quantity identifiers.

## 2.7    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 and RANGE. These properties have a semantics that is equivalent to that of the corresponding YAGO relations. RDFS also knows SUBCLASSOF and SUBPROPERTYOF (i.e. SUBRELATIONOF) as well, but these relations are not acyclic as they are in YAGO. Acyclicity is crucial for YAGO, but RDFS does not know the concept of an acyclic transitive relation. This entails that the property ATR can be defined and used, but that RDFS would not know its semantics.

RDFS also knows reification, i.e. it can express facts about facts. To refer to such a fact, RDFS does not use fact identifiers, but it characterizes the fact by giving its relation and its arguments. For example, to talk about the fact (Elvis, BORNINYEAR,1935), RDFS would create a new entity for the fact (say, elvisFact) and characterize it as follows:

| elvisFact | RDF:RELATION | BORNINYEAR |
| elvisFact | RDF:SUBJECT | Elvis |
| elvisFact | RDF:OBJECT | 1935 |
| elvisFact | RDF:TYPE | statement |

This process is called *reifying the fact*. Then, elvisFact can be used as an argument in other facts. Different from the YAGO model, though, the reified fact does not become part of the ontology – let alone the model. In RDFS, arbitrary facts can be used as arguments, even ones that are false in the model.

Thus, to model YAGO's reification, one would need to reify each fact of the ontology in the above manner so that each fact is present both in the ontology and as a reified fact. To simplify this process, the XML syntax of RDFS allows *triple identifiers*. If a fact of the ontology is equipped with a triple identifier, that fact is automatically reified. This allows us to map a YAGO ontology into RDFS. The following excerpt shows how the sample fact of Section 2.3 can be represented in RDFS. Each fact of YAGO becomes a triple in RDFS with a triple identifier.

```
<rdf:Description rdf:about="http://mpii.de/yago#Elvis">
  <yago:hasWonPrize rdf:ID="f1"
   rdf:resource="http://mpii.de/yago#GrammyAward" />
</rdf:Description>
<rdf:Description rdf:about="http://mpii.de/yago#f1">
  <yago:inYear rdf:ID="f2">1967</yago:inYear>
</rdf:Description>
```

YAGO uses fact identifiers, but it does not have built-in relations to make logical assertions about facts (e.g. it does not allow saying 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 ATR. However, this is about to change. OWL is currently being refined to its successor, OWL 1.1[117]. The extended description logic $\mathcal{SROIQ}$ [76], which has been adopted as the logical basis of OWL 1.1, allows expressing irreflexivity and transitivity. This allows defining acyclic transitivity, even though SUBCLAS-SOF and SUBPROPERTYOF remain reflexive and transitive and hence not acyclic. We plan to investigate the relation of YAGO and OWL, once OWL 1.1 has been fully established.

## 2.8 Query Language

To demonstrate the use of YAGO, we present a query language for reification graphs. A *pattern* for a reification graph $G_{N,I,L}$ over a set of variables $V, V \cap (N \cup I \cup L) = \emptyset$, is a reification graph over the set of nodes $N \cup V$, the set of identifiers $I \cup V$ and the set of labels $L \cup V$. A *matching* of a pattern $P$ for

a graph $G$ is a substitution $\sigma : V \to N \cup I \cup L$, such that $\sigma(P) \subset G$. $\sigma(P)$ is called a *match*.

Our syntax simplifications from section 2.2 can be transferred to patterns: Each implicit fact identifier becomes a fresh variable. Thus, e.g., the query *"When did Elvis win the Grammy Award?"* can be formulated as

<div align="center">

Elvis HASWONPRIZE Grammy Award INYEAR \$x

</div>

which is shorthand for

```
$i1: Elvis   HASWONPRIZE   Grammy Award
$i2: $i1     INYEAR        $x
```

A match of that pattern for the ontology would map the variables to entities such that the pattern becomes a subgraph of the ontology.

Usually, each entity that appears in the query also has to appear in the ontology. If that is not the case, there is no match. However, we may want to allow a query such as *"Which singers were born after 1930?"*, even if 1930 does not appear in the ontology. We cannot simply add all existing literals to the YAGO ontology because a YAGO ontology has to be finite. Hence, we introduce *filter relations* (such as AFTER), which are not part of the match, but are evaluated on the match as filters. Technically, a filter relation is a decidable function that maps two literals to either 0 or 1. Then, a *filter pattern* $P$ for a reification graph $G_{N,I,R}$ over a set of literals $L$, a set of variables $V, V \cap (N \cup I \cup R \cup L) = \emptyset$ and a set of filter relations $F$, is a reification graph over the set of nodes $N \cup V \cup L$, the set of identifiers $I \cup V$ and the set of labels $R \cup V \cup F$. For example, the following is a filter pattern over the set of literals $\{1930\}$ and the set of filter relations $\{\text{AFTER}\}$:

```
$i1: $x   TYPE        singer
$i2: $x   BORNINYEAR  $y
$i3: $y   AFTER       1930
```

A *matching for a filter pattern* is a matching $\sigma$ for the pattern $P \setminus \{(i, (a_1, r, a_2)) | r \in F\}$, such that $\forall (i, (a_1, r, a_2)) \in P, r \in F : r(\sigma(a_1), \sigma(a_2)) = 1$. In the example, a matching would have to bind \$x and \$y in such a way that AFTER(\$y, 1930) = 1. \$i3 is left unbound. Then, a *match for a filter pattern* is a matching applied to the pattern, i.e. in our case e.g.

```
#1:   Elvis   TYPE    singer
#2:   Elvis   BORNINYEAR   1935
$i3:  1935    AFTER   1930
```

See Section 4.4 for implementation issues.

# 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[66]. 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 3.0, WordNet contains 82,115 synsets for 117,798 unique nouns. (Wordnet also includes other types of words like verbs and adjectives, but we consider only nouns in this article.) 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 article, we focus on hypernyms/hyponyms. Conceptually, the hypernymy relation in WordNet spans a directed acyclic graph (DAG) with a single root 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 November 2007, which comprised 2,000,000 articles at that time. Each Wikipedia article is a single Web page and usually describes a single topic or entity.

The majority of Wikipedia pages have been manually assigned to one or multiple *categories*. The page about Elvis Presley, for example, is in the categories *American rock singers*, *1935 births*, and 34 more.

Furthermore, a Wikipedia page may have an *infobox*. An infobox is a standardized table with information about the entity described in the article. For example, there is a standardized infobox for people, which contains the

birth date, the profession, and the nationality. Other widely used infoboxes exist for cities, music bands, companies etc.

For our information extraction, we use the XML dump of Wikipedia. It is approximately 3 Gigabytes large and stores the articles in the original wiki markup language.

# 4 Information Extraction

The construction of the YAGO ontology takes place in two stages: First, different heuristics are applied to Wikipedia to extract candidate entities and candidate facts. This stage also establishes the connection between Wikipedia and WordNet. Then, quality control techniques are applied. We will now explain these steps in detail and then explain how YAGO is stored.

## 4.1 Wikipedia Heuristics

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 `Albert Einstein` in our ontology. The page titles in Wikipedia are unique. Our algorithm parses the XML dump of Wikipedia and applies 4 different types of heuristics to the articles.

### 4.1.1 Infobox Heuristics

**Attributes and Values**

A Wikipedia article may contain an infobox (see Figure 2). An infobox is a rich source of facts about the article entity. Each row in the infobox table contains an attribute and a value. For example, an infobox on the page of Elvis Presley may contain the attribute *Born* with the value *January 8, 1935*. We have identified 170 highly frequent attributes. For each of these attributes, we have manually designed a YAGO relation, the *target relation*. For example, for the attribute *Born*, we introduced the relation BIRTHDATE with domain `person` and range `timeInterval`. Some attributes use the same relation. For example, both *Born* and *Birthday* map to the relation BIRTH-DATE. In principle, each row of the infobox will generate one fact. Its first argument is the article entity, its relation is determined by attribute and its

second argument is the value of the attribute. However, we map some attributes to the inverse of a relation. For example, the attribute *official name* has as its value the official name of the article entity. But instead of generating the fact (*entity*, HASOFFICIALNAME, *officialname*), our algorithm rather generates the fact (*officialname*, MEANS, *entity*). For the purpose of the knowledge extraction, we call these attributes *inverse attributes*.

Figure 2: A Wikipedia Infobox



| **Elvis Presley** | |
|---|---|
| Elvis in 1970 | |
| **Background information** | |
| **Birth name** | Elvis Aaron Presley[1] |
| **Also known as** | Elvis |
| **Born** | January 8, 1935 <br> Tupelo, Mississippi |
| **Origin** | Memphis, Tennessee |
| **Died** | August 16, 1977 (aged 42) <br> Memphis, Tennessee |
| **Genre(s)** | Rockabilly, Rock and Roll, Gospel, Blues, Country |
| **Occupation(s)** | Singer, Actor |
| **Instrument(s)** | Vocals, Guitar, Piano |
| **Years active** | 1954–1977 |
| **Label(s)** | Sun, RCA |
| **Website** | Elvis.com |

24

Some attributes may have multiple values. For example, a person may have multiple children. In this case, one row of the infobox will generate multiple facts – one HASCHILD fact for each child. Again other attributes do not concern the article entity, but another fact. For example, the attribute *GDPasOf* gives the year in which the gross domestic product (GDP) of a country was computed. In this case, the algorithm does not generate the fact (*country*, GDPASOF,*year*), but rather the fact (*id*,DURING,*year*), where *id* is the id of the previously established fact (*country*,HASGDP,*gdp*). Thus, we get the following fact (in shorthand notation)

$$country \quad \text{HASGDP} \quad gdp \quad \text{DURING} \quad year$$

Sometimes, the meaning of the attribute depends on the type of infobox. For example, the *length* of a car is an extent in space, whereas the *length* of a song is a duration. Hence we allow ambiguous attributes to be qualified by the type of the infobox (in this example we distinguish car infoboxes and song infoboxes). In summary, an *infobox heuristics* is a manually established mapping from a (possibly qualified) attribute to the target relation that stores whether the attribute is an inverse attribute, whether it allows multiple values and whether it is about another fact.

**Parsing**

When our algorithm finds an infobox, it walks through all of its attributes. If a heuristics is available for the attribute, the algorithm tries to parse the value of the attribute as an instance of the range of the target relation. For example, the attribute *Birth date* has the target relation BIRTHDATE. Its range is `timeInterval`. Hence the parser tries to parse the value of the attribute as a time interval (i.e. as a year or a date expression). We use the parsers from [138] to parse literals of different types. This parser uses regular expressions to parse numbers, dates and quantities. It also normalizes units of measurement to ISO units. If the range of the target relation is not a literal class (but, e.g. the class `person`), the parser expects a Wikipedia entity as value and hence tries to find a Wikipedia link. If the parse fails, the attribute is ignored. Inverse attributes and attributes with multiple values are handled accordingly. Last, the type of the infobox (e.g. *city infobox* or *person infobox*) produces a candidate fact that establishes the article entity as an instance of the respective class.

There is one exception: For each country, Wikipedia contains a page on its economy (e.g. a page with the title "Economy of the United States"). In

these cases, the parser is configured to attach the extracted facts not to an entity `economy of the United States` but rather to the country itself.

## 4.1.2 Type Heuristics

### Wikipedia Categories

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*).

### 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 [138]). 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 [138] 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.

### The Wikipedia Category Hierarchy

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, Elvis is in the category *Grammy Awards*). Thus, the hierarchy is of little use from an ontological point of view. Hence we take only the leaf categories of Wikipedia and ignore all higher categories. Then we use

WordNet to establish the hierarchy of classes, because WordNet offers an ontologically well-defined taxonomy of synsets.

## 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.

## Connecting Wikipedia and WordNet

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. These approaches lead to non-satisfactory results.

Finally, we found that the following solution works best: 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 around two 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 Word-Net 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 Word Heuristics

**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 a class for each synset known to WordNet (i.e. `city`). Second, we establish a MEANS relation between each word of synset and the corresponding class (i.e. ("metropolis", MEANS, `city`)).

**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. Each redirect gives us one 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 [138] to identify and decompose the person names.

### 4.1.4 Category Heuristics

**Relational Categories**

Relational Wikipedia categories give valuable information about the article entity. For example, if a page is in the category *Rivers in Germany*, then we know that the article entity is LOCATEDIN Germany. Category information is very useful, because not every article has an infobox, but most articles

have categories. We designed simple *category heuristics* to exploit the category names. Each heuristics is basically a pair of a regular expression (e.g. "Mountains|Rivers in (.*)") and a target relation (e.g. LOCATEDIN). If a category name matches the regular expression, a new fact is added, where the first argument is the article entity, the relation is the target relation and the second argument is the string captured by the brackets of the regular expression. If, e.g., the *Rhine* is in the category *Rivers in Germany*, then we add the fact (Rhine,LOCATEDIN,Germany). Table 1 shows some of our category heuristics.

Table 1: Some Category Heuristics

| Regular Expression | Relation |
| --- | --- |
| ([0-9]{3,4}) births | BORNONDATE |
| ([0-9]{3,4}) deaths | DIEDONDATE |
| ([0-9]{3,4}) establishments | ESTABLISHEDONDATE |
| ([0-9]{3,4}) books\|novels | WRITTENONDATE |
| Mountains\|Rivers in (.*) | LOCATEDIN |
| Presidents\|Governors of (.*) | POLITICIANOF |
| (.*) winners | HASWONPRIZE |
| [A-Za-z]+ (.*) winners | HASWONPRIZE |

Since all candidate facts will be type checked, we can be generous with our heuristics. For example, the last two heuristics will extract "American Nobel Prize" and "Nobel Prize", respectively, from the category name "American Nobel Prize winners". Of course, "Nobel Prize" is the correct choice, because the category says that the prize winner is American, not the prize. At this stage, however, we keep both candidates and rely on the type check to sort out the wrong one (see Section 4.2.2).

**Language Categories**

There are some special categories that indicate the name of the article entity in other languages. For example, the city of London is in the special category *fr:Londres*, meaning that London is called "Londres" in French. Our algorithm maps the language prefix "fr" to the appropriate language entity (French) and adds the following candidate fact:

London ISCALLED "*Londres*" INLANGUAGE French

## 4.2 Quality Control

Our goal is to deliver an ontology of high quality. For this purpose, we developed rigorous quality control mechanisms. *Canonicalization* makes each fact and each entity reference unique. As a result, an entity is always referred to by the same identifier in all facts in YAGO. *Type Checking* eliminates individuals that do not have a class. It also eliminates facts that do not respect the domain and range constraints of their relation. As a result, an argument of a fact in YAGO is always an instance of the class required by the relation. We will now discuss these steps in detail.

### 4.2.1 Canonicalization

**Redirect Resolution**

Our infobox heuristics deliver facts that have Wikipedia entities (i.e. Wikipedia links) as arguments. These links, however, need not be the correct Wikipedia page identifiers. For example, a reference to the city of `Saint Petersburg` may be given as the link *St. Petersburg*. If one clicks on that link, Wikipedia's redirect system will seamlessly forward to the correct page *Saint Petersburg*, but for our ontology, these incorrect links have to be resolved. So, for each argument of each candidate fact, our algorithm checks whether the argument is an incorrect Wikipedia identifier and replaces it by the correct, redirected, Wikipedia identifier.

**Removal of Duplicate Facts**

Sometimes, two heuristics deliver the same fact. In this case, our canonicalization eliminates one of them. Furthermore, if one fact is more precise than another, then only the more precise fact is kept. For example, if the category heuristics has determined a birth date of `1935` and the infobox heuristics has determined `1935-01-08`, then only the fact with `1935-01-08` is kept.

### 4.2.2 Type Checking

**Reductive Type Checking**

A candidate fact may contain an entity for which the heuristics could not determine its class. Since we cannot validate such a fact, our algorithm discards these facts. The same applies to Wikipedia entities that have been proposed for an article, but that do not have a page yet. For the remaining facts, our algorithm knows the class(es) and all super classes for each entity. If it encounters a fact where the first argument is not in the domain of the

31

relation, this fact is eliminated (similarly for the second argument and the range). This type constraint also applies to literals, but the heuristics already make sure that literals have the correct data type.

**Inductive Type Checking**

Type constraints can be used not only to eliminate facts, but also to generate facts. If, for example, some entity has a birth date, then one could infer that the entity is a person – rather than eliminating the fact due to lack of type information. We call this process *inductive type checking*, as opposed to reductive type checking. We have made the experience that for people entities, inductive type checking works very well. So whenever a fact contains an unknown entity and the range or domain of the relation predicts that the entity should be a person, the algorithm keeps the fact and makes the entity an instance of the class `person`. Reductive type checking is not applied in these cases. We use a regular expression check to make sure that the entity name follows the basic pattern of given name and family name.

## 4.3  Storage

**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 with the DESCRIBES relation. 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**

When 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 USING relation to identify the technique by which a fact was extracted and the DURING relation to give the time of the extraction. 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.

**File Format**

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. With each fact, we store the estimated accuracy as a value between 0 and 1 (as given by our evaluation, see Section 5). 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. We also provide an RDFS version of YAGO, as explained in Section 2.7. Furthermore, YAGO can be converted to a database table. The table has the simple schema

$$FACTS(factId, arg1, relation, arg2, accuracy)$$

We provide software to load YAGO into an Oracle, Postgres, or MySQL database.

## 4.4 Query Engine

We implemented a simple query engine along the lines of [87] on top of the database version of YAGO. It can solve queries of the form described in Section 2.8. The engine first normalizes the shorthand notations to the standard notation, so that each line of the query consists of a fact identifier, a first argument, a relation and a second argument. Since entities can have several names in YAGO, we have to deal with ambiguity. Our query engine makes sure that each word in the query is considered in all of its possible meanings. For this purpose, we replace each non-literal, non-variable argument in the query by a fresh variable and add a MEANS fact for it. We call this process *word resolution*. Consider, for example, the query *"Who was born after Elvis?"*:

```
$i1: Elvis   BORNONDATE   $e
$i2: $x      BORNONDATE   $y
$i3: $y      AFTER   $e
```

This query becomes

```
$i0: "Elvis"   MEANS   $Elvis
$i1: $Elvis   BORNONDATE   $e
```

```
$i2:  $x        BORNONDATE    $y
$i3:  $y        AFTER   $e
```

An answer to this query shall bind the variables of the original, non-normalized query (assume them to be `$e`, `$x` and `$y`) and the variables introduced by the word resolution (i.e. in our case `$Elvis`). We first discard lines with filter relations. In our example, the last line is discarded. Then, one single SQL query is fired. It contains one SELECT argument for each variable that we want to bind and one join for each line of the query. In the example, the SQL query is

```
SELECT f0.arg2, f1.arg2, f2.arg1, f2.arg2
FROM facts f0, facts f1, facts f2
WHERE f0.arg1='"Elvis"'
AND f0.relation='means'
AND f1.arg1=f0.arg2
AND f1.relation='bornOnDate'
AND f2.relation='bornOnDate'
```

This query delivers values for the variables `$Elvis`, `$e`, `$x` and `$y`. Then, the query engine evaluates the AFTER relation on the pair `$y`/`$e`. If the relation holds, the binding of the variables is returned as a result.

Technically, each individual is an instance of all superclasses of its class in the deductive closure of the YAGO ontology. To avoid computing the deductive closure, we introduced the pseudo-relation ISA. It holds between an entity and all classes that the entity is an instance of. The query engine resolves an ISA relation to a sequence of queries, which try several path lengths up to a certain limit. For example, the query

<div align="center">

Elvis   ISA   $x

</div>

is converted to the following sequence of queries

```
$i1:  Elvis   TYPE   $x


$i1:  Elvis   TYPE   $x1
$i2:  $x1     SUBCLASSOF    $x


$i1:  Elvis   TYPE   $x1
$i2:  $x1     SUBCLASSOF    $x2
```

<div align="center">34</div>

```
$i3: $x2      subclassOf   $x
...
```

This implementation leaves much room for improvement, especially concerning efficiency. For example, it takes several seconds to return 5 answers to the query *"Who was born after Elvis?"*. Queries with more joins, especially with ISA and with MEANS, can take several minutes. In this article, we use the engine only to showcase the contents of YAGO.

# 5 Evaluation

## 5.1 Precision

We were interested in the precision of YAGO. To evaluate the precision 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 human judges and asked them to assess whether the facts were correct. For each fact, judges could click "correct", "incorrect" or "don't know". Since common sense often does not suffice to judge the correctness of YAGO facts, we also presented them 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 some small fraction of 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 or FOUNDIN. This is why we evaluated only those facts that stem from a heuristics. 13 judges participated in the evaluation and evaluated a total number of 5200 facts. We report the precision of the most precise and least precise heuristics groups in Table 2. To be sure that our findings are significant, we computed the Wilson interval [23] for $\alpha = 5\%$. A precision of 100% means that all facts produced by the heuristics have been evaluated exhaustively.

The evaluation shows very good results. 74 heuristics have a precision of over 95%. Especially the crucial link between WordNet and Wikipedia, *WordNetLinker*, turned out to be very accurate. Also, the use of conceptual categories (*ConceptualCategory*) and infobox types (*InfoboxType*) to establish the TYPE relation proved very fruitful. *establishedInCat* is a category heuristics, the other heuristics shown in the table are infobox heuristics.

Table 2: Precision of YAGO's heuristics

|  | Heuristics | #Eval | Precision | |
|---|---|---|---|---|
| 1 | hasExpenses | 46 | 100.0 % | ± 0.0 % |
| 2 | hasInflation | 25 | 100.0 % | ± 0.0 % |
| 3 | hasLaborForce | 43 | 97.67441% | ± 0.0 % |
| 4 | during | 232 | 97.48950% | ± 1.838 % |
| 5 | ConceptualCategory | 59 | 96.94342% | ± 3.056 % |
| 6 | participatedIn | 59 | 96.94342% | ± 3.056 % |
| 7 | plays | 59 | 96.94342% | ± 3.056 % |
| 8 | establishedInCat | 57 | 96.84294% | ± 3.157 % |
| 9 | createdOn | 57 | 96.84294% | ± 3.157 % |
| 10 | originatesFrom | 57 | 96.84294% | ± 3.157 % |
| | ... | | | |
| 72 | WordNetLinker | 56 | 95.11911% | ± 4.564 % |
| | ... | | | |
| 74 | InfoboxType | 76 | 95.08927% | ± 4.186 % |
| 75 | hasSuccessor | 53 | 94.86150% | ± 4.804 % |
| | ... | | | |
| 88 | hasGDPPPP | 75 | 91.22189% | ± 5.897 % |
| 89 | hasGini | 62 | 91.00750% | ± 6.455 % |
| 90 | discovered | 84 | 90.98286% | ± 5.702 % |

Our algorithms cannot always achieve a precision of 100%. One reason for this is purely statistical: even if all of our assessed sample facts are correct (as they were indeed for many heuristics), the center of the Wilson interval will be lower than 100% to account for the uncertainty that is inherent in a confidence estimation. Some fraction of the assessed facts was extracted incorrectly. For example, the inductive type checking mistook a racing horse for a person, because it had a birth date. The WordNetLinker made the Los Angeles Angels of Anaheim managers a subclass of angel. Another source of error are inconsistencies of the underlying sources. For example, for the relation bornOnDate, most false facts stem from erroneous Wikipedia categories (e.g. some person born in 1802 is in the Wikipedia category *1805 births*). For facts with literals (such as hasHeight), many errors stem from a non-standard format of the numbers (giving, e.g., one movie actor the

height of 1.6km, just because the infobox says *1,632m* instead of *1.632m*).
Occasionally, the data in Wikipedia was updated between the time of our
extraction and the time of the evaluation. This explains many errors in
HASGDPPPP and HASGINI. In addition, the evaluation of an ontology is
sometimes a philosophical issue, because even simple relations suffer from
vagueness. For example, 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? These cases of disputability are 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 ap-
proaches, because the approaches usually differ in the choice of relations
and in the choice of the sources. Furthermore, precision can usually be var-
ied at the cost of recall. Approaches that use pattern matching (e.g. the
Espresso System [116] or LEILA [137]) typically achieve precision rates of
50% to 92%, depending on the extracted relation. State-of-the-art taxonomy
induction as described in [128] achieves a precision of 84%. KnowItAll [65]
and KnowItNow [27] are reported to have precision rates of 85% and 80%,
respectively. TextRunner [13] is able to extract a large amount of facts (11.3
million) out of which only an estimated 69% (7.8 million) are well-formed.
Of these well-formed facts, the authors estimate that 82% are correct. Wu et
al. [148] aim at filling in missing values in Wikipedia infoboxes and achieve
a remarkable precision of 73% to 97%. Ponzetto et al. [118] exploit the
Wikipedia category network to construct a taxonomy and achieve an preci-
sion of around 87%. Banko et al. [15] use different domain search strategies
for fact extraction and show an precision of around 80%.

## 5.2   Size

Table 3 shows the number of entities in YAGO. The overall number of entities
is 1.7 million.

Table 3: Number of entities in YAGO

| Relations | 92 |
|---|---|
| Classes | 224,391 |
| Individuals (without words and literals) | 1,531,588 |

Table 4 shows the number of facts for the most frequent relations in YAGO. The overall number of ontological facts is 15 million. This number does not include the respective witness facts (FOUNDIN, DURING and USING) and the trivial facts (INUNIT, HASVALUE and DESCRIBES). YAGO profits most from the infoboxes about movies, persons, and geopolitical entities.

Table 4: Largest relations in YAGO

| Relation | # Facts | Relation | # Facts |
|---|---|---|---|
| hasUTCOffset | 12724 | hasWonPrize | 13645 |
| livesIn | 15185 | writtenInYear | 16441 |
| originatesFrom | 16876 | directed | 18633 |
| hasPredecessor | 19154 | actedIn | 22249 |
| hasDuration | 23652 | bornInLocation | 24400 |
| hasImdb | 24659 | hasArea | 26781 |
| hasProductionLanguage | 27840 | produced | 30519 |
| hasPopulation | 30731 | isOfGenre | 33898 |
| hasSuccessor | 46658 | establishedOnDate | 69529 |
| hasWebsite | 79779 | created | 83627 |
| locatedIn | 125738 | diedOnDate | 168037 |
| subClassOf | 211979 | bornOnDate | 350613 |
| givenNameOf | 464816 | familyNameOf | 466969 |
| inLanguage | 2389627 | isCalled | 2984362 |
| type | 3957223 | means | 4014819 |

It is not easy to compare the size of YAGO to other ontologies, because the ontologies usually differ in their structure, their types of axioms, their relations, their domain, and their quality. For informational purposes, we list the current number of entities and facts for some of the most important other domain-independent ontologies in Table 5, as given on the respective Web sites. DBpedia is huge, but it includes YAGO.

39

Table 5: Size of other ontologies

| Ontology | # Entities | # Facts |
|---|---|---|
| SUMO [114] | 20,000 | 60,000 |
| Ponzetto et al. [118] | n/a | 110,000 |
| WordNet [66] | 117,659 | 821,492 |
| Cyc [106] | 300,000 | 3,000,000 |
| TextRunner [13] | n/a | 7,800,000 |
| YAGO | 1,700,000 | 15,000,000 |
| DBpedia [9] | 1,950,000 | 103,000,000 |

# 6 Applications

## 6.1 Querying

As described in Section 4.4, we have implemented a query engine for accessing the content of YAGO. Table 6 shows two simple queries on the ontology. The second query makes use of the distinction between words and other individuals in YAGO.

Table 6: Simple queries on YAGO

| Query | Result |
|---|---|
| Who was Einstein's doctoral advisor?<br>Einstein HASDOCTORALADVISOR `$x` | `$x=Alfred Kleiner` |
| Who is named after a place in Africa?<br>`$place` locatedin Africa<br>`$name` means `$place`<br>`$name` familynameof `$who` | `$who=Gabriel Sudan`<br>and 22 more |

Table 7 shows three advanced queries. The first query uses a virtual relation ($>$) to ask for countries having a higher Human Development Index (HDI) than Canada. YAGO knows 5. The other queries show how reified facts work.

Table 7: Advanced queries on YAGO

| | |
|---|---|
| Which countries have a higher HDI than Canada?<br><br>Canada HASHDI $HDIcanada<br>$other HASHDI $HDIother<br>$HDIother > $HDIcanada | $other=Sweden<br>and 4 others |
| When did Angela Merkel become chancellor?<br>Angela Merkel ISA chancellor<br>    SINCE $when | $when=2005-11-22 |
| How is Germany called in Italian?<br>Germany ISCALLED $how<br>    INLANGUAGE Italian | $how="Germania" |

It is tempting to assume some kind of "completeness" of YAGO and to ask, e.g. how to say a particular word in Italian, who governed a particular country at a particular point of time or who was a particular person's doctoral advisor. It should not be forgotten, however, that YAGO cannot know more than what is available in the infoboxes and categories of Wikipedia. YAGO's knowledge is huge, but it cannot be complete.

## 6.2 Other Uses

Notwithstanding its young age, YAGO has already found several applications in different areas of research.

**Semantic Search**

YAGO is the basis for the semantic search engines NAGA [87] and ESTER [16]. NAGA utilizes YAGO as a knowledge base for graph-based information retrieval. It allows querying YAGO in a SPARQL-like fashion and ranks the answers according to their "prominence". Its ranking mechanism uses YAGO's data model to formalize notions like the compactness, informativeness and confidence of answer graphs. ESTER combines full text search and ontological search by weaving the YAGO ontology into a text corpus. This allows ESTER to deliver hybrid answers that incorporate both data from the text and from the ontology.

**Entity Ranking**

Stoyanovich et al.[132] build an enriched Web graph, which contains Web pages and the entities mentioned in them. Based on this graph, the authors propose authority-based ranking techniques that combine Web page authorities and entity authorities into a mutual reinforcement process. The ontological basis for the enriched graph structure is YAGO.

**Disambiguation**

Demartini [51] aims at finding per-topic experts among the Wikipedia authors. YAGO's semantics is exploited to refine and disambiguate Wikipedia topics in the expert finding process.

**Knowledge Acquisition**

The idea of YAGO's category heuristics has been applied by Ponzetto et al. [118] to extract ontological knowledge from Wikipedia's category system. YAGO itself has been integrated into the DBpedia project [9], where it forms the taxonomic backbone of the knowledge base.

# 7 Conclusion

## 7.1 Summary

We presented our ontology YAGO and the methodology for constructing it automatically. We explained the logical model behind YAGO and showed how it extends the data model of RDFS to represent $n$-ary relations. We proved that, despite the expressiveness of the model, its consistency is still decidable. Furthermore, we could show that the model allows computing a unique smallest base for any given YAGO ontology.

We showed how the category system and the infoboxes of Wikipedia can be exploited for knowledge extraction. We explained how Wikipedia and WordNet can be linked and how we can enforce high precision through type checks.

Our evaluation showed not only that YAGO is one of the largest knowledge bases available today, but also that it has an unprecedented quality in the league of automatically generated ontologies.

## 7.2 Discussion

Although the knowledge extraction itself runs in a fully automated way, a one-time manual effort was necessary to bootstrap the extraction. We identified and defined attributes and relations for the infoboxes, and we established the patterns for the category heuristics. Furthermore, we manually identified some exceptions for the heuristics that connects Wikipedia and WordNet. Given the huge amount of knowledge that we could extract in return and given the high precision of the data that we could achieve, we believe that the manual effort was justified.

So far, YAGO's extraction mechanisms are tailored to Wikipedia and WordNet. However, our work has created a rich framework of methods that can be applied to other sources as well. Many sources, such as the catalogue of Amazon.com or the Internet Movie Database, use category systems and

structures that are similar to infoboxes. Furthermore, techniques such as inductive and reductive type checking can be applied in other scenarios, too. Finally, YAGO itself can be useful for other information extraction projects, e.g., to check the plausibility of the extracted facts.

## 7.3   Outlook

YAGO opens up new opportunities and challenges. On the theoretical side, we plan to investigate how the YAGO model and OWL 1.1 can be reconciled, once OWL 1.1 has been fully developed. Furthermore, the efficiency of the query engine deserves attention. On the practical side, we plan to enrich YAGO by further facts from other sources. We hope that future information extraction can profit from the knowledge that YAGO already provides – for example to do type checks or to generate seed pairs. This could result in a positive feedback loop, in which the addition of knowledge helps the extraction of new knowledge.

YAGO can be freely downloaded from our Web site `http://www.mpii.mpg.de/yago`. 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

Let $\mathcal{F}$ be a (finite) set of fact triples, as defined in Chapter 2.4. Let $\to$ be the rewrite system defined there (see [11] for a reference on term rewriting). All rules of the rewrite system are of the form $F \to F \cup \{f\}$, where $F \subseteq \mathcal{F}$ and $f \in \mathcal{F}$. Hence $\to$ is monotone. Furthermore, $\mathcal{F}$ is finite. Hence $\to$ is finitely terminating. It is easy to see that if $F \to F \cup \{f_1\}$ and $F \to F \cup \{f_2\}$ for some $F \subseteq \mathcal{F}$ and $f_1, f_2 \in \mathcal{F}$, then

$$F \to F \cup \{f_1\} \to F \cup \{f_1, f_2\}$$
$$F \to F \cup \{f_2\} \to F \cup \{f_1, f_2\}$$

Hence $\to$ is locally confluent. Since $\to$ is finitely terminating, $\to$ is globally confluent and convergent. Thus, given any set of facts $F \subseteq \mathcal{F}$, the largest set $D_F$ with $F \to^* D_F$ 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 2.4.

LEMMA 1: [*No circular rules*]
*Let $y$ be a consistent YAGO ontology, and $\{f_1, ..., f_n\}$ 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*

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

**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 $(r, \text{TYPE}, \texttt{acyclicTransitive-Relation}) \in D(y)$ and hence $y$ cannot be consistent.

47

- $n = 2$, 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 $(r, \text{TYPE}, \texttt{acyclicTransitiveRelation}) \in D(y)$ and hence $y$ cannot be consistent.

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

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

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

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 $D(y)\backslash\{f\} \to D(y)$, there must be a rule $G \hookrightarrow f$ with $G \subseteq D(y)\backslash\{f\}$. Let $i \in [0, n]$ be 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:** Let $b$ be a canonical base of a consistent YAGO ontology $y$. Let $B = range(b)$. We define the set

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

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. Scaling information extraction to large document collections. *IEEE Data Eng. Bull.*, 28(4), 2005.

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

[3] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[4] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[5] F. Alexander. A parser for hpsg, 1990.

[6] S. Amer-Yahia, P. Case, T. Rölleke, J. Shanmugasundaram, and G. Weikum. Report on the db/ir panel at sigmod 2005. *SIGMOD Record*, 34(4):71–74, 2005.

[7] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. The MIT Press, 2004.

[8] J. Aslam and S. Decatur. On the sample complexity of noise-tolerant learning. *Information Processing Letters*, 57(4):189–195, 1996.

[9] 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*, volume 4825 of *LNCS*, pages 722–735. Springer, 2007.

[10] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In *Mechanizing Mathematical Reasoning*, pages 228–248, 2005.

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

[12] R. I. Bainbridge. Montagovian definite clause grammar. In *EACL*, pages 25–34, Geneva, Switzerland, 1985.

[13] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, pages 2670–2676, 2007.

[14] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, 2007.

[15] M. Banko and O. Etzioni. Strategies for lifelong knowledge extraction from the web. In *K-CAP '07: Proceedings of the 4th international conference on Knowledge capture*, pages 95–102, New York, NY, USA, 2007. ACM.

[16] H. Bast, A. Chitea, F. M. Suchanek, and I. Weber. Ester: efficient search on text, entities, and relations. In *SIGIR*, pages 671–678, 2007.

[17] P. Baumgartner and F. M. Suchanek. Automated reasoning support for first-order ontologies. In J. Alferes, J. Bailey, W. May, and U. Schwertel, editors, *PPSWR*, volume 4187 of *LNAI*. Springer, 2006.

[18] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.

[19] W. Bidgood. The snomed dicom microglossary, 1998.

[20] P. V. Biron and A. Malhotra. Xml schema part 2: Datatypes second edition. `http://www.w3.org/TR/xmlschema-2/#built-in-datatypes`.

[21] S. Brin. Extracting patterns and relations from the world wide web. In *Selected papers from the Int. Workshop on the WWW and Databases*, pages 172–183, London, UK, 1999. Springer-Verlag.

[22] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[23] L. D. Brown, T. T. Cai, and A. DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001.

[24] P. Buitelaar, D. Olejnik, and M. Sintek. A protege plug-in for ontology extraction from text based on linguistic analysis. In *ESWS*, Heraklion, Greece, 2004.

[25] P. Buitelaar and S. Ramaka. Unsupervised ontology-based semantic tagging for knowledge markup. In W. Buntine, A. Hotho, and S. Bloehdorn, editors, *Workshop on Learning in Web Search at the ICML*, 2005.

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

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

[28] M. J. Cafarella, C. Re, D. Suciu, and O. Etzioni. Structured querying of web text data: A technical challenge. In *CIDR*, pages 225–234, 2007.

[29] M. Califf and R. Mooney. Relational learning of pattern-match rules for information extraction. *Workshop in Natural Language Learning at ACL*, pages 9–15, 1997.

[30] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufman, San Francisco, USA, 2003.

[31] S. Chakrabarti. Breaking through the syntax barrier: Searching with entities and relations. In *PKDD*, 2004.

[32] S. Chakrabarti. Breaking through the syntax barrier: Searching with entities and relations. In *ECML*, pages 9–16, 2004.

[33] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW*, pages 571–580, 2007.

[34] 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.

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

[36] T. Cheng, X. Yan, and K. C.-C. Chang. Entityrank: Searching entities directly and holistically. In *VLDB*, pages 387–398, 2007.

[37] T. Cheng, X. Yan, and K. C.-C. Chang. Entityrank: Searching entities directly and holistically. In *VLDB*, 2007.

[38] V. Cherkassky and M. Yunqian. Practical selection of svm parameters and noise estimation for svm regression. *Neural Networks*, 17(1):113–126, 2004.

[39] P. Cimiano, G. Ladwig, and S. Staab. Gimme the context: Contextdriven automatic semantic annotation with cpankow. In A. Ellis and T. Hagino, editors, *WWW*, Chiba, Japan, 2005.

[40] P. Cimiano and J. Vlker. Text2onto - a framework for ontology learning and data-driven change discovery, 2005.

[41] P. Cimiano and J. Völker. Towards large-scale, open-domain and ontology-based named entity classification. In *RANLP*, pages 166–172, 2005.

[42] F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *IJCAI*, Seattle, USA, 2001.

[43] S. Cohen, Y. Kanza, B. Kimelfeld, and Y. Sagiv. Interconnection semantics for keyword search in xml. In *CIKM*, pages 389–396, 2005.

[44] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. Xsearch: A semantic search engine for xml. In *VLDB*, pages 45–56, 2003.

[45] W. Cohen. Information extraction. http://www.cs.cmu.edu/ wcohen/ie-survey.ppt, 2004.

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

[47] M. Collins. Three generative, lexicalised models for statistical parsing. In *ACL*, pages 16–23, Morristown, NJ, USA, 1997. Association for Computational Linguistics.

[48] B. Crysmann, A. Frank, B. Kiefer, S. Mueller, G. Neumann, J. Piskorski, U. Schaefer, M. Siegel, H. Uszkoreit, F. Xu, M. Becker, and H. U. Krieger. An integrated architecture for shallow and deep processing. In *ACL*, pages 441–448, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[49] H. Cunningham. *An Introduction to Information Extraction*. Elsevier, 2005.

[50] 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.

[51] G. Demartini. Finding experts using wikipedia. In A. V. Zhdanova, L. J. B. Nixon, M. Mochol, and J. Breslin, editors, *Proceedings of the Workshop on Finding Experts on the Web with Semantics (FEWS2007) at ISWC/ASWC2007, Busan, South Korea*, November 2007.

[52] P. DeRose, W. Shen, F. C. 0002, A. Doan, and R. Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *VLDB*, pages 399–410, 2007.

[53] P. DeRose, W. Shen, F. Chen, Y. Lee, D. Burdick, A. Doan, and R. Ramakrishnan. Dblife: A community information management platform for the database research community (demo). In *CIDR*. www.crdrdb.org, 2007.

[54] J. Diederich and W.-T. Balke. The semantic growbag algorithm: Automatically deriving categorization systems. In *ECDL*, pages 1–13, 2007.

[55] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. In *WWW*, pages 178–186. ACM Press, 2003.

[56] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1), 2006.

[57] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1):64–72, 2006.

[58] A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Managing information extraction: state of the art and research directions. In *SIGMOD*, 2006.

[59] A. Doms, T. Furche, A. Burger, and M. Schroeder. How to query the geneontology. In *Knowledge Representation in Bioinformatics*, 2005.

[60] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. In *WWW*, 2006.

[61] J. Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, 1970.

[62] N. K. (Editor). Special issue on data management issues in social sciences. *IEEE Data Eng. Bull.*, 30(2), 2007.

[63] O. Etzioni, M. Banko, and M. J. Cafarella. Machine reading. In *AAAI*, 2006.

[64] O. Etzioni, M. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *AI*, 165(1), 2005.

[65] 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.

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

[67] A. Finn and N. Kushmerick. Multi-level boundary classification for information extraction. In *ECML*, pages 111–122, 2004.

[68] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *AAAI*, 2000.

[69] N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas, editors. *Workshop of the INitiative for the Evaluation of XML Retrieval (INEX), Schloss Dagstuhl, Germany, December 9-11, 2002*, 2002.

[70] J. Graupmann. Concept-based search on semi-structured data exploiting mined semantic relations. In *EDBT Workshops*, pages 34–43, 2004.

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

[72] R. V. Guha, R. McCool, and E. Miller. Semantic search. In *WWW*, pages 700–709, 2003.

[73] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD*, pages 16–27, 2003.

[74] J. Halpern. *Reasoning about Uncertainty*. MIT Press, 2005.

[75] A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *ICCL*, Nantes, France, 1992.

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

[77] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, 2002.

[78] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[79] 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.

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

[81] F. C. J. Iria. Relation extraction for mining the semantic web, 2005.

[82] R. Jaschke, L. B. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in folksonomies. In *PKDD*, pages 506–514, 2007.

[83] T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 29(1):40–48, 2006.

[84] T. Joachims. *Learning to Classify Text Using Support Vector Machines*. PhD thesis, University of Dortmund, 2002.

[85] B. H. K. C-C. Chang and Z. Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR*, 2005.

[86] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

[87] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *ICDE*. IEEE, 2008.

[88] G. Kasneci, F. M. Suchanek, M. Ramanath, and G. Weikum. How NAGA Uncoils: Searching with Entities and Relations. In *WWW*, New York, NY, USA, 2007. ACM Press.

[89] M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. In S. J. Hanson, G. A. Drastal, and R. L. Rivest, editors, *Computational Learning Theory and Natural Learning Systems.* Bradford/MIT Press, 1994.

[90] B. Kimelfeld and Y. Sagiv. Efficient engines for keyword proximity search. In *WebDB*, pages 67–72, 2005.

[91] D. Kinzler. Wikisense - mining the wiki. In *Wikimania*, 2005.

[92] D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. In *NIPS*, 2002.

[93] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *ACM*, 46(5):604–632, 1999.

[94] D. Konopnicki and O. Shmueli. Database-inspired search. In *VLDB*, page 2, 2005.

[95] W.-S. Li, K. S. Candan, Q. Vu, and D. Agrawal. Retrieving and organizing web pages by "information unit". In *WWW*, pages 230–244, 2001.

[96] D. Lin and P. Pantel. Dirt: Discovery of inference rules from text. In *KDD*, pages 323–328, 2001.

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

[98] J. Lloyd. *Foundations of Logic Programming.* Springer, 1987.

[99] G. Luo, C. Tang, and Y. li Tian. Answering relationship queries on the web. In *WWW*, pages 561–570, 2007.

[100] J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, and C. Yu. Navigating the seas of structured web data. In *CIDR*, 2007.

[101] A. Maedche and S. Staab. Discovering conceptual relations from text. In W. Horn, editor, *ECAI*, pages 85–94, Berlin, Germany, 2000.

[102] A. Maedche and S. Staab. Learning ontologies for the semantic web. In *WWW*, Honkong, China, 2001.

[103] G. Mann and D. Yarowsky. Multi-field information extraction and cross-document fusion. In *ACL*, 2005.

[104] C. D. Manning and H. Schutze. *Foundations of Statistical NLP*. MIT Press, Cambridge, MA, USA, 1999.

[105] I. Marshall and . Sfr. Extraction of semantic representations from syntactic cmu link grammar linkages. In G. A. et al, editor, *RANLP*, pages 154–159, Tzigov Chark, Bulgaria, 2001.

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

[107] J. Maxwell and R. Kaplan. An efficient parser for lfg. In *LFG Conf.*, 1996.

[108] B. McLernon and N. Kushmerick. Transductive pattern learning for information extraction. In *Proc. Workshop Adaptive Text Extraction and Mining at EACL*, 2006.

[109] D. Milne, I. H. Witten, and D. Nichols. A knowledge-based search engine powered by wikipedia. In *ACM Conference on Information and Knowledge Management (CIKM)*, 2007.

[110] R. Montague. Universal grammar. In *Formal Philosophy. Selected Papers of Richard Montague.* Yale University Press, 1974.

[111] Z. Nie, Y. Ma, S. Shi, J.-R. Wen, and W.-Y. Ma. Web object retrieval. In *WWW*, pages 81–90, 2007.

[112] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: bringing order to web objects. In *WWW*, 2005.

[113] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: bringing order to web objects. In *WWW*, pages 567–574, 2005.

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

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

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

[117] P. F. Patel-Schneider and I. Horrocks. Owl 1.1 web ontology language. http://www.w3.org/Submission/owl11-overview/.

[118] S. P. Ponzetto and M. Strube. Deriving a large-scale taxonomy from wikipedia. In *AAAI*, pages 1440–1445, 2007.

[119] D. Ravichandran and E. Hovy. Learning surface text patterns for a question answering system. In *ACL*, Philadelphia, USA, 2002.

[120] E. Riloff. Automatically generating extraction patterns from untagged text. *Annual Conf. on AI*, pages 1044–1049, 1996.

[121] E. Rosch, C. Mervis, W. Gray, D. Johnson, and P. Boyes-Bream. Basic objects in natural categories. *Cognitive Psychology*, pages 382–439, 1976.

[122] M. Ruiz-Casado, E. Alfonseca, and P. Castells. Automatic assignment of wikipedia encyclopedic entries to wordnet synsets. In *AWIC*, pages 380–386, 2005.

[123] 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.

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

[125] A. Schutz and P. Buitelaar. Relext: A tool for relation extraction in ontology extension. In Y. Gil, E. Motta, E. Benjamins, and M. Musen, editors, *ISWC*, pages 593–606. Springer, 2005.

[126] H. U. Simon. General bounds on the number of examples needed for learning probabilistic concepts. In *COLT*, pages 402–411, New York, NY, USA, 1993. ACM Press.

[127] D. Sleator and D. Temperley. Parsing english with a link grammar. *3rd Int. Workshop on Parsing Technologies*, 1993.

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

[129] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, pages 233–272, 1999.

[130] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. Crystal: Inducing a conceptual dictionary. *IJCAI*, pages 1314–1319, 1995.

[131] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

[132] J. Stoyanovich, S. Bedathur, K. Berberich, and G. Weikum. Entityauthority: Semantically enriched graph-based authority propagation. In *Proceedings of 10th International Workshop on Web and Databases (WebDB 2007)*, page n/a, Beijing, China, 2007. n/a.

[133] M. Strube and S. P. Ponzetto. WikiRelate! Computing Semantic Relatedness Using Wikipedia. In *AAAI*, 2006.

[134] F. Suchanek, G. Ifrim, and G. Weikum. Combining linguistic and statistical analysis to extract relations from web documents. Research Report MPI-I-2006-5-004, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, March 2006.

[135] 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.

[136] F. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. Research Report MPI-I-2006-5-006, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, 2006.

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

[138] 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.

[139] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *WWW*, New York, NY, USA, 2007. ACM Press.

[140] M. Theobald, R. Schenkel, and G. Weikum. Efficient and self-tuning incremental query expansion for top-k query processing. In *SIGIR*, pages 242–249, New York, NY, USA, 2005. ACM Press.

[141] M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for topx search. In *VLDB*, pages 625–636, 2005.

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

[143] O. Udrea, D. Recupero, and V. S. Subrahmanian. Annotated rdf. In *ESWC*, pages 487–501, 2006.

[144] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*. W. H. Freeman & Co., New York, NY, USA, 1990.

[145] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic wikipedia. In *WWW*, 2006.

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

[147] N. Weber and P. Buitelaar. Web-based ontology learning with isolde. In *Proc. of ISWC2006 Workshop on Web Content Mining with Human Language Technologies*, 2006.

[148] F. Wu and D. S. Weld. Autonomously semantifying wikipedia. In *CIKM*, 2007.

[149] F. Xu and H. U. Krieger. Integrating shallow and deep nlp for information extraction. In *RANLP*, Borovets, Bulgaria, 2003.

[150] F. Xu, D. Kurz, J. Piskorski, and S. Schmeier. Term extraction and mining term relations from free-text documents in the financial domain. In *Int. Conf. on Business Information Systems*, Poznan, Poland, 2002.

[151] R. Yangarber, R. Grishman, P. Tapanainen, and S. Huttunen. Automatic acquisition of domain knowledge for information extraction. In *ICCL*, pages 940–946, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[152] R. Yangarber, W. Lin, and R. Grishman. Unsupervised learning of generalized names. In *ICCL*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[153] L. Zhang and Y. Yu. Learning to generate CGs from domain specific sentences. *LNCS*, 2120:44–57, 2001.

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

.

| MPI-I-2007-5-003 | Fabian M. Suchanek, Gjergji Kasneci, Gerhard Weikum | Yago: A Large Ontology from Wikipedia and WordNet |
| --- | --- | --- |
| MPI-I-2007-RG1-002 | T. Hillenbrand, C. Weidenbach | Superposition for Finite Domains |
| MPI-I-2007-5-002 | K. Berberich, S. Bedathur, T. Neumann, G. Weikum | A Time Machine for Text Search |
| MPI-I-2007-5-001 | G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum | NAGA: Searching and Ranking Knowledge |
| MPI-I-2007-4-006 | C. Dyken, G. Ziegler, C. Theobalt, H. Seidel | GPU Marching Cubes on Shader Model 3.0 and 4.0 |
| MPI-I-2007-4-005 | T. Schultz, J. Weickert, H. Seidel | A Higher-Order Structure Tensor |
| MPI-I-2007-4-004 | C. Stoll | A Volumetric Approach to Interactive Shape Editing |
| MPI-I-2007-4-003 | R. Bargmann, V. Blanz, H. Seidel | A Nonlinear Viseme Model for Triphone-Based Speech Synthesis |
| MPI-I-2007-4-002 | T. Langer, H. Seidel | Construction of Smooth Maps with Mean Value Coordinates |
| MPI-I-2007-4-001 | J. Gall, B. Rosenhahn, H. Seidel | Clustered Stochastic Optimization for Object Recognition and Pose Estimation |
| MPI-I-2007-2-001 | A. Podelski, S. Wagner | A Method and a Tool for Automatic Veriication of Region Stability for Hybrid Systems |
| MPI-I-2007-1-002 | E. Althaus, S. Canzar | A Lagrangian relaxation approach for the multiple sequence alignment problem |
| MPI-I-2007-1-001 | E. Berberich, L. Kettner | Linear-Time Reordering in a Sweep-line Algorithm for Algebraic Curves Intersecting in a Common Point |
| MPI-I-2006-5-006 | G. Kasnec, F.M. Suchanek, G. Weikum | Yago - A Core of Semantic Knowledge |
| 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 $\mathbb{R}^3$ |
| 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 | A. Efremov, R. Mantiuk, K. Myszkowski, H. Seidel | Design and Evaluation of Backward Compatible High Dynamic Range Video Compression |
| 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 | H. Bast, I. Weber, C.W. Mortensen | Output-Sensitive Autocompletion Search |
| MPI-I-2006-1-006 | M. Kerber | Division-Free Computation of Subresultants Using Bezout Matrices |
| MPI-I-2006-1-005 | A. Eigenwillig, L. Kettner, N. Wolpert | Snap Rounding of Bézier Curves |
| MPI-I-2006-1-004 | S. Funke, S. Laue, R. Naujoks, L. Zvi | Power Assignment Problems in Wireless Communication |
| 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 |
| MPI-I-2005-4-005 | G. Krawczyk, M. Goesele, H. Seidel | Photometric Calibration of High Dynamic Range Cameras |
| MPI-I-2005-4-004 | C. Theobalt, N. Ahmed, E. De Aguiar, G. Ziegler, H. Lensch, M.A. Magnor, H. Seidel | Joint Motion and Reflectance Capture for Creating Relightable 3D Videos |
| MPI-I-2005-4-003 | T. Langer, A.G. Belyaev, H. Seidel | Analysis and Design of Discrete Normals and Curvatures |
| MPI-I-2005-4-002 | O. Schall, A. Belyaev, H. Seidel | Sparse Meshing of Uncertain and Noisy Surface Scattered Data |
| MPI-I-2005-4-001 | M. Fuchs, V. Blanz, H. Lensch, H. Seidel | Reflectance from Images: A Model-Based Approach for Human Faces |
| MPI-I-2005-2-004 | Y. Kazakov | A Framework of Refutational Theorem Proving for Saturation-Based Decision Procedures |
| MPI-I-2005-2-003 | H.d. Nivelle | Using Resolution as a Decision Procedure |
| MPI-I-2005-2-002 | P. Maier, W. Charatonik, L. Georgieva | Bounded Model Checking of Pointer Programs |
| MPI-I-2005-2-001 | J. Hoffmann, C. Gomes, B. Selman | Bottleneck Behavior in CNF Formulas |
| MPI-I-2005-1-008 | C. Gotsman, K. Kaligosi, K. Mehlhorn, D. Michail, E. Pyrga | Cycle Bases of Graphs and Sampled Manifolds |
| MPI-I-2005-1-007 | I. Katriel, M. Kutz | A Faster Algorithm for Computing a Longest Common Increasing Subsequence |
| MPI-I-2005-1-003 | S. Baswana, K. Telikepalli | Improved Algorithms for All-Pairs Approximate Shortest Paths in Weighted Graphs |
| MPI-I-2005-1-002 | I. Katriel, M. Kutz, M. Skutella | Reachability Substitutes for Planar Digraphs |
| MPI-I-2005-1-001 | D. Michail | Rank-Maximal through Maximum Weight Matchings |
| MPI-I-2004-NWG3-001 | M. Magnor | Axisymmetric Reconstruction and 3D Visualization of Bipolar Planetary Nebulae |
| MPI-I-2004-NWG1-001 | B. Blanchet | Automatic Proof of Strong Secrecy for Security Protocols |
| MPI-I-2004-5-001 | S. Siersdorfer, S. Sizov, G. Weikum | Goal-oriented Methods and Meta Methods for Document Classification and their Parameter Tuning |
| MPI-I-2004-4-006 | K. Dmitriev, V. Havran, H. Seidel | Faster Ray Tracing with SIMD Shaft Culling |
| MPI-I-2004-4-005 | I.P. Ivrissimtzis, W.-. Jeong, S. Lee, Y.a. Lee, H.-. Seidel | Neural Meshes: Surface Reconstruction with a Learning Algorithm |