

Automated Construction and Growth of a Large Ontology

Fabian M. Suchanek

Thesis for obtaining the title of Doctor of Engineering
of the Faculties of Natural Sciences and Technology
of Saarland University

Saarbrücken, Germany, 2009-01-12

Dean: Prof. Dr. Joachim Weickert
Faculty of Mathematics and Computer Science
Saarland University
Saarbrücken, Germany

Colloquium: 2008-12-19
Max-Planck-Institute for Informatics
Saarbrücken, Germany

Examination Board

Supervisor and First Reviewer: Prof. Dr.-Ing. Gerhard Weikum
Department for Databases and Information Systems
Max-Planck Institute for Informatics
Saarbrücken, Germany

Second Reviewer: Prof. Dr. Rudi Studer
Institute of Applied Informatics
and Formal Description Methods
Karlsruhe University (TH)
Karlsruhe, Germany

Third Reviewer: Prof. Dr. Amit Sheth
Kno.e.sis Center
Wright State University
Dayton, Ohio, USA

Chairman: Prof. Dr. Andreas Zeller
Department for Computer Science
Saarland University
Saarbrücken, Germany

Research Assistant: Dr. Ralf Schenkel
Department for Databases and Information Systems
Max-Planck Institute for Informatics
Saarbrücken, Germany

Abstract

An ontology is a computer-processable collection of knowledge about the world. This thesis explains how an ontology can be constructed and expanded automatically. The proposed approach consists of three contributions:

1. A core ontology, YAGO.
YAGO is an ontology that has been constructed automatically. It combines high accuracy with large coverage and serves as a core that can be expanded.
2. A tool for information extraction, LEILA.
LEILA is a system that can extract knowledge from natural language texts. LEILA will be used to find new facts for YAGO.
3. An integration mechanism, SOFIE.
SOFIE is a system that can reason on the plausibility of new knowledge. SOFIE will assess the facts found by LEILA and integrate them into YAGO.

Each of these components comes with a fully implemented system. Together, they form an integrative architecture, which does not only gather new facts, but also reconcile them with the existing facts. The result is an ever-growing, yet highly accurate ontological knowledge base. A survey of applications of the ontology completes the thesis.

Kurzfassung

Eine Ontologie ist eine Wissenssammlung über die Welt, die von einem Computer verarbeitet werden kann. Die vorliegende Dissertation beschreibt, wie eine Ontologie automatisch erstellt und erweitert werden kann. 3 Komponenten werden dafür vorgestellt:

1. Eine Kern-Ontologie, YAGO.
YAGO ist eine Ontologie, die automatisch erstellt wurde. Sie beinhaltet bereits eine große Menge an Fakten mit hoher Genauigkeit. YAGO dient als Ausgangs-Ontologie, die es zu erweitern gilt.
2. Ein System zur Informationsextraktion, LEILA.
LEILA ist eine Software, die Informationen aus natürlichsprachigen Texten extrahieren kann. LEILA wird benutzt, um neue Fakten für YAGO zu finden.
3. Ein Integrations-Mechanismus, SOFIE.
SOFIE ist ein System, das herausfinden kann, ob ein neuer Fakt plausibel ist. SOFIE wird die Fakten, die LEILA gefunden hat, beurteilen und sie in YAGO integrieren.

Jede dieser Komponenten ist vollständig implementiert. Zusammen bilden sie eine integrierte Architektur, die nicht nur neue Fakten sammeln kann, sondern sie auch mit den bereits gesammelten Fakten vereinigen kann. Das Ergebnis ist eine stetig wachsende Ontologie von hoher Genauigkeit. Die Dissertation schließt mit einem Überblick über die Anwendungen der Ontologie.

Summary

An ontology is a computer-processable collection of knowledge about the world. This thesis explains how an ontology can be constructed and expanded automatically. The proposed approach consists of three contributions: The first, YAGO, is a core ontology. The second, LEILA, is a system that can extract information from text documents. The third, SOFIE, is an integration mechanism that weaves the information found by LEILA into YAGO.

YAGO. YAGO [135, 136] is the core ontology. It currently knows 2 million entities. This includes thousands of famous people, as well as cities, historic events, and movies. It knows more than 19 million facts about them. For example, YAGO knows the following things:

<i>ElvisPresely</i>	<i>is a</i>	<i>singer</i>
<i>singer</i>	<i>subClassOf</i>	<i>person</i>
<i>ElvisPresely</i>	<i>bornOnDate</i>	<i>1935-01-08</i>
<i>ElvisPresely</i>	<i>bornIn</i>	<i>Tupelo</i>
<i>Tupelo</i>	<i>locatedIn</i>	<i>Mississippi(state)</i>
<i>Mississippi(state)</i>	<i>locatedIn</i>	<i>USA</i>

YAGO has been constructed automatically. Its knowledge has been extracted from one of the most comprehensive lexicons available today, Wikipedia. The facts from Wikipedia have been combined with the taxonomic structure of WordNet [59]. The key contributions of YAGO are the following:

1. **Information Extraction from Wikipedia.** The YAGO approach builds on the *infoboxes* and *category pages* in Wikipedia. We present techniques that harvest the infoboxes and category pages in a systematic manner, yielding millions of highly accurate facts.
2. **Combination with WordNet.** We explain how the information from Wikipedia can be linked to the information from WordNet [59]. 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.
3. **Quality Control.** We explain how we can enforce the high accuracy of our extraction through *type checking*. Type checking leverages the information that has already been extracted to verify the plausibility of newly extracted data.

YAGO comes with a novel knowledge representation model. It extends RDFS and is powerful enough to reify facts. Nevertheless, we can prove that its consistency is decidable. Our evaluation shows 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.

LEILA. LEILA [133, 134] is system that can extract facts from text documents. LEILA finds patterns in the text that express a certain relation. For example, LEILA can find out that the pattern “*X was born in Y*” indicates that someone was born in a certain place. By harvesting these patterns systematically, LEILA

can extract, for example, all pairs of people with their birth places from a text corpus. LEILA brings three key contributions:

1. **Linguistic Analysis:** LEILA uses a link-grammar representation [122] for natural-language sentences. This allows LEILA to detect robust natural language patterns.
2. **Counterexamples:** LEILA takes into account counterexamples, i.e., information that is known to be wrong. This allows LEILA to identify and discard erroneous patterns.
3. **Machine Learning:** LEILA uses statistical learning to generalize the useful patterns. This process gives LEILA high yield and robust patterns.

Our theoretical analysis proves that erroneous patterns cannot disrupt the performance of LEILA. Their influence converges to zero as the corpus size grows. Our evaluation shows that LEILA outperforms state-of-the-art techniques for information extraction.

SOFIE. It is not trivial to add the facts found by LEILA to the ontology YAGO. For example, one needs to make sure that the new facts do not contradict the existing ones. This is the task accomplished by SOFIE. SOFIE is a novel model of information integration, which casts the task of information extraction into a logical reasoning problem. In this model, word disambiguation, pattern matching, and rule-based reasoning on the ontology all become part of one unified framework. More precisely, SOFIE combines three capabilities in a single system:

1. **Word Disambiguation:** If a word has multiple meanings, the system can figure out the most likely meaning in a text. Different from existing systems, however, SOFIE will automatically re-consider its choice if more evidence for another meaning becomes available.
2. **Pattern Matching:** SOFIE extends LEILA and finds patterns in text documents to extract facts. Unlike other systems, it can reason on the plausibility of patterns and reject patterns if counter evidence becomes available.
3. **Ontological Reasoning:** SOFIE makes full use of the background knowledge from YAGO. SOFIE can take into account constraints on the relations, links between hypotheses and connections to the existing knowledge.

We show how the unifying model of SOFIE can be cast into a weighted MAX SAT problem. We explain how the MAX SAT problem can be solved efficiently in our case, and we prove an approximation guarantee. Our evaluation shows that SOFIE can extract clean, canonicalized facts even from arbitrary, unstructured Web documents with a precision of over 90%.

A survey about applications that use YAGO already completes the thesis.

Zusammenfassung

Eine Ontologie ist eine Wissenssammlung über die Welt, die von einem Computer verarbeitet werden kann. Die vorliegende Dissertation beschreibt, wie eine Ontologie automatisch erstellt und erweitert werden kann. 3 Komponenten werden dafür vorgestellt: Die erste Komponente, YAGO, ist eine Kern-Ontologie, die es auszubauen gilt. Die zweite Komponente, LEILA, ist ein System, das Informationen aus Text-Dokumenten extrahieren kann. Die dritte Komponente, SOFIE, ist ein Integrationsmechanismus, der die von LEILA gefundenen Informationen in YAGO einfügen kann.

YAGO. YAGO [135, 136] ist die Kern-Ontologie. Sie umfasst momentan mehr als 2 Millionen Entitäten. Dies schließt bekannte Leute des öffentlichen Lebens ebenso ein wie Städte, historische Ereignisse, Filme und vieles mehr. YAGO weiß mehr als 19 Millionen Fakten über die Entitäten. YAGO weiß zum Beispiel die folgenden Dinge:

<i>ElvisPresely</i>	<i>ist ein</i>	<i>Sänger</i>
<i>Sänger</i>	<i>Unterklasse von</i>	<i>Person</i>
<i>ElvisPresely</i>	<i>geboren am</i>	<i>1935-01-08</i>
<i>ElvisPresely</i>	<i>geboren in</i>	<i>Tupelo</i>
<i>Tupelo</i>	<i>liegt in</i>	<i>Mississippi(state)</i>
<i>Mississippi(state)</i>	<i>liegt in</i>	<i>USA</i>

YAGO wurde automatisch erzeugt. YAGO bringt drei Neuheiten:

1. **Informationsextraktion aus Wikipedia.** Mit YAGO können die *Infoboxen* und das *Kategoriensystem* von Wikipedia systematisch ausgenutzt werden. Dies liefert Millionen von hochgradig akkuraten Fakten.
2. **Kombination mit WordNet.** Die Information aus Wikipedia wird mit der Information aus WordNet [59] kombiniert. Dadurch umfasst YAGO sowohl die große Anzahl von Individuen aus Wikipedia, als auch die saubere Taxonomie aus WordNet.
3. **Qualitäts-Kontrolle.** Die hohe Akkuratheit der YAGO-Ontologie wird durch *Typ-Checks* sichergestellt. Typ-Checks nutzen die Information, die bereits extrahiert wurde, um die Plausibilität von neu extrahierter Information einzuschätzen.

YAGO nutzt ein neuartiges Modell zur Wissensrepräsentation. Dieses Modell erweitert RDFS und ist mächtig genug, um Fakten zu reifizieren. Trotzdem ist die Konsistenz des Modells entscheidbar. Eine Evaluation beweist, dass YAGO nicht nur eine der größten frei verfügbaren Wissensdatenbanken ist, sondern auch eine nie da gewesene Qualität aufweist.

LEILA. LEILA [133, 134] ist ein System das Fakten aus Text-Dokumenten extrahieren kann. LEILA findet Wortfolgen, die bestimmte Bedeutungen haben. Zum Beispiel kann LEILA herausfinden, dass die Wortfolge "*X wurde in Y*

geboren” auf den Geburtsort von X hinweist. Durch systematisches Ausnutzen dieser Wortfolgen kann LEILA beispielsweise alle Paare aus Personen mit ihrem Geburtsort aus einem Text-Korpus extrahieren. LEILA bringt drei Neuheiten:

1. **Linguistische Analyse:** LEILA untersucht die Sätze im Korpus grammatikalisch [122]. Dadurch kann LEILA bedeutungstragende Wortfolgen viel zuverlässiger identifizieren.
2. **Gegenbeispiele:** LEILA benutzt Gegenbeispiele, also Informationen, von denen bekannt ist, dass sie falsch sind. Dadurch kann LEILA irreführende Wortfolgen leichter entdecken und ausschließen.
3. **Maschinelles Lernen:** LEILA benutzt statistisches Lernen um sinnvolle Wortfolgen zu verallgemeinern. Dadurch findet LEILA robuste und ertragsreiche Wortfolgen.

Eine theoretische Untersuchung beweist, dass irreführende Wortfolgen LEILA nicht aus der Bahn werfen können. Ihr Einfluss konvergiert gegen Null wenn die Korpusgröße wächst. Eine ausführliche Evaluation zeigt, dass LEILA Informationen besser extrahieren kann als andere Ansätze.

SOFIE. Es ist nicht einfach, die Fakten, die LEILA findet, in YAGO zu integrieren. Zum Beispiel muss man darauf achten, dass die neu hinzugewonnenen Fakten sich nicht mit den alten widersprechen. Diese Aufgaben wird von SOFIE übernommen. SOFIE ist ein neuartiges Modell zur Informationsintegration, welches Informationsextraktion als logisches Problem auffasst. In SOFIE werden die Disambiguierung von Wörtern, das Untersuchen von Wortfolgen und das Beachten semantischer Regeln alle Teil eines vereinheitlichten Modells. SOFIE vereint drei Prozesse in einem System:

1. **Disambiguierung:** Wenn ein Wort mehrere Bedeutungen hat, so kann SOFIE herausfinden, was das Wort in einem bestimmten Zusammenhang wohl bedeutet. Anders als existierende Systeme kann SOFIE diese Entscheidung auch wieder rückgängig machen, wenn etwas darauf hinweist, dass das Wort doch in einer anderen Bedeutung gemeint ist.
2. **Untersuchung von Wortfolgen:** SOFIE erweitert LEILA und findet Wortfolgen in Text-Dokumenten. Anders als existierende Systeme kann SOFIE über die Plausibilität von Wortfolgen nachdenken und vormalig sinnvoll erscheinende Wortfolgen auch wieder verwerfen.
3. **Ontologisches Reasoning:** SOFIE nutzt das vorhandene Hintergrundwissen von YAGO voll aus. SOFIE kann semantische Randbedingungen, Zusammenhänge zwischen Fakten und Zusammenhänge zwischen altem und neuem Wissen beachten.

Dieses vereinheitlichte Modell kann als MAX SAT Problem interpretiert werden. Die vorliegende Arbeit zeigt, wie dieses Problem effizient und mit einer Approximationsgarantie gelöst werden kann. Eine Evaluation zeigt, dass SOFIE selbst aus unstrukturierten Internetdokumenten saubere Fakten extrahieren kann – mit einer Präzision von über 90%.

Die Dissertation schließt mit einem Überblick über die Anwendungen der Ontologie.

Acknowledgements

This work would not have been possible without the persistent support of my supervisor Prof. Dr.-Ing. Gerhard Weikum. I would like to thank him for his scientific guidance, for his always inspiring input, and, most of all, for his ability to constantly radiate optimism and good humor. Furthermore, I would like to thank my colleagues and friends Georgiana Ifrim and Gjergji Kasneci, with whom I had a wonderful time here at the Max Planck Institute in the “Semantic Office”. Many other people with whom I have worked deserve my thanks, among them Mauro Sozio, Maya Ramanath, Gerard de Melo, Holger Bast and Milan Vojnović from Microsoft Research Cambridge. Finally, I would like to thank the German Research Foundation DFG for granting me financial support. The DFG gave me complete freedom in the choice of my research. This allowed me to combine my aims in research with my aims in life: the exploration of existing knowledge and the discovery of new knowledge.

Contents

1	Introduction	15
1.1	Motivation	15
1.2	Contribution	16
1.3	Philosophical Background	16
1.3.1	Epistemology	17
1.3.2	Ontology	19
2	Knowledge Representation	23
2.1	RDFS/OWL	23
2.1.1	Symbols	23
2.1.2	Statements	24
2.1.3	Predefined Symbols	24
2.1.4	Reification	26
2.1.5	Extension to OWL	26
2.1.6	Difficulties with the RDFS/OWL model	27
2.2	The YAGO Model	27
2.2.1	Informal Description	27
2.2.2	Reification Graphs	28
2.2.3	n -ary Relations	29
2.2.4	Data Types	30
2.2.5	Semantics	31
2.2.6	Reification and Semantics	34
2.2.7	Relation to RDFS/OWL	35
2.2.8	Query Language	36
2.3	Summary	37
3	YAGO	39
3.1	Overview	39
3.1.1	Problem Statement	39
3.1.2	Related Work	40
3.1.3	Contribution	42
3.1.4	Sources for YAGO	43
3.2	Construction of YAGO	46
3.2.1	Information Extraction	46
3.2.2	Quality Control	51
3.2.3	Storage and Export Formats	52
3.2.4	Query Engine	54
3.3	Evaluation and Demonstration	55

3.3.1	Precision	55
3.3.2	Size	58
3.3.3	Demonstration of Querying Capabilities	59
3.4	Conclusion	60
4	LEILA	61
4.1	Overview	61
4.1.1	Problem Statement	61
4.1.2	Related Work	62
4.1.3	Contribution	64
4.1.4	Linguistic Analysis	64
4.2	System Model	66
4.2.1	Preprocessing	66
4.2.2	Algorithm	67
4.2.3	Robustness	70
4.2.4	Classifying Patterns	73
4.3	Experiments	76
4.3.1	Setup	76
4.3.2	Results	79
4.4	Conclusion	82
5	SOFIE	85
5.1	Overview	85
5.1.1	Problem statement	85
5.1.2	Related Work	86
5.1.3	Contribution	92
5.2	Model	93
5.2.1	Statements	93
5.2.2	Rules	95
5.2.3	MAX-SAT Model	98
5.3	Implementation	101
5.3.1	Adaptation of LEILA	101
5.3.2	Weighted MAX SAT Algorithm	106
5.3.3	SOFIE Algorithm	113
5.4	Experiments	114
5.4.1	Information Extraction from Semi-Structured Text	114
5.4.2	Information Extraction from Web Documents	120
5.4.3	Comparison of Weighted MAX SAT Algorithms	127
5.5	Conclusion	129
6	Applications	131
6.1	NAGA	131
6.1.1	Ontological Search and Ranking	131
6.1.2	Query Language	132
6.1.3	Ranking	133
6.2	ESTER	135
6.2.1	Ontological Search and Full-Text Search	135
6.2.2	Prefix Search Engines	135
6.2.3	Weaving the Ontology into the Text	136
6.3	TOB	138

6.4	Tagbooster User Study	139
6.4.1	Social Tagging	139
6.4.2	Meaningfulness	140
6.4.3	Influence of Suggestions	142
6.5	Other Applications	143
6.6	Summary	145
7	Conclusion	147
A	YAGO Detailed Data	149
A.1	Attribute Map	149
A.2	Relations	154
A.3	Heuristics	157
B	Proofs	161
B.1	Convergence of \rightarrow	161
B.2	Uniqueness of the Canonical Base	161
B.3	Bound on False Labels	163
B.4	Approximation Guarantee of FMS	165
B.5	Simple MAX SAT Algorithm	166
B.6	Safe Variables	167
B.7	Approximation Guarantee of FMS*	168

Chapter 1

Introduction

1.1 Motivation

The Quest For Understanding. Computers were invented half a century ago. They have become a valuable and sometimes indispensable help in many areas of our everyday lives. Still, computers could be even more helpful if they had a certain knowledge about the world. If they understood English, for example, they could communicate with us in natural language. If they understood the meaning of words, they could also perform better in automated text translation. If they understood what we are searching for on the Internet, they could better help us finding the desired bit of information. In order to be of use for a computer, the knowledge about the world would have to be encoded in a machine-readable formalism. Such a formal collection of world knowledge is called an *ontology*¹ [129].

The Use of Ontologies. Wherever ontologies are already available, they are heavily used. This applies above all to applications in the vision of the Semantic Web, but also to numerous other application fields: Ontological knowledge is used for tasks such as machine translation [86, 34, 104], word sense disambiguation [27, 46, 72], and document classification [74, 15, 127]. It is also employed for information integration [93, 103, 16], in particular for data cleaning and record linkage (entity resolution) [44, 35, 84]. It is used to analyze the behavior of semantic systems [137] and for tasks such as question answering [141, 19, 73] and query expansion [88, 66, 139]. In addition, there are emerging trends towards entity- and fact-oriented Web search and community management [12, 29, 33, 38, 51, 81, 89, 97, 100], which can build on rich ontological knowledge bases.

Challenges. The construction of an ontology is a non-trivial task. Knowledge has to be explored, gathered and formalized. This alone is a tedious enterprise, since an ontology usually contains thousands, if not millions of facts. Furthermore, an ontology has to be steadily kept up to date. New knowledge emerges

¹In this thesis, we use the term *ontology* in a very general sense to include also facts about instances. See Section 1.3.2 for a discussion.

day by day and has to be combined with the existing knowledge. For these reasons, ontology construction is still a highly active area of research.

1.2 Contribution

This thesis presents a novel approach for ontology construction. It allows not only constructing an ontology, but also automatically enlarging it by new knowledge. Our approach consists of three contributions:

1. A core ontology, YAGO.
YAGO is an ontology that has been constructed automatically. It combines high accuracy with large coverage and serves as a core that can be expanded.
2. A tool for information extraction, LEILA.
LEILA is a system that can extract knowledge from natural language texts. LEILA will be used to find new facts for YAGO.
3. An integration mechanism, SOFIE.
SOFIE is a system that can reason on the plausibility of new knowledge. SOFIE will assess the facts found by LEILA and integrate them into YAGO.

Each of these components comes with a fully implemented system. Together, they form an integrative architecture, which does not only gather new facts, but also reconcile them with the existing facts. The result is an ever-growing, yet highly accurate ontological knowledge base. This thesis will show how background knowledge, information extraction and information integration can work together seamlessly: The more knowledge is already present, the easier it is to interpret the new knowledge. For example, once our system has learnt that Elvis Presley is dead, it knows that all living people who claim to be Elvis must be epigones.

Outline. The rest of this work is structured as follows: The remainder of this chapter will introduce the basic concepts of Epistemology and Ontology. The second chapter will discuss how knowledge can be represented in a computer. The third, fourth and fifth chapter form the main part of the thesis: They will introduce the new ontology, the new fact gathering tool, and the integration mechanism, respectively. Each of these chapters also discusses related work in detail, often examining the same reference from different points of view. The sixth chapter will show where our ontology is already employed, and the seventh chapter concludes this thesis.

1.3 Philosophical Background

This section will set up the philosophical background for our knowledge gathering system. It will present and illustrate the puzzles in the areas of Epistemology and Ontology. It cannot solve these puzzles, but it will list the assumptions that are necessary to prepare the ground for semantic computer systems. The section will also define the notions of *knowledge* and *ontology*. It will do so first from a

psychological and philosophical point of view. Then, it will show in which sense these notions can be transferred to computer science.

1.3.1 Epistemology

Epistemology. Epistemology is the philosophical study of the nature of knowledge. It is concerned with the concepts of belief and truth. It analyzes what knowledge can be acquired, how knowledge can be acquired and what it means to acquire knowledge. The field is immensely complex and brings up numerous puzzles, traps and pitfalls. To overcome them, we will introduce a number of simplifying assumptions. These assumptions will allow us to see reality as a set of true statements for the sake of this thesis.

Knowledge. In general, one distinguishes two types of knowledge: *declarative knowledge* and *procedural knowledge*. Declarative knowledge concerns information expressed by statements, such as the information that Paris is in France. Procedural knowledge concerns abilities to perform certain tasks, such as the ability to ride a bicycle. There is some evidence that these two types of knowledge work through different psychological processes in the human mind [119]. In this thesis, we will be concerned only with declarative knowledge, i.e., with knowledge in the form of statements.

Beliefs. A *belief* is the psychological state in which a person holds a statement to be true². There are different theories about the nature of beliefs: Representationalism assumes that the statement is stored in some way in the human mind. This view is challenged by more behavioristic schools of thought, which argue that a person's behavior is the only observable variable and hence the most important key to somebody's beliefs. Other philosophers question the existence of beliefs altogether, opining that the notion of belief is nothing more than a folk-psychological metaphor for a phenomenon that science has not yet explored³.

Since we are dealing with computer systems in this thesis, we will adapt a black box view on this matter. For a computer system, we will simply see *belief* as a function from statements to truth values. We will say that the system *believes* a statement, if the function evaluates to *true* for that statement. Adopting Plato's formulation of knowledge as "true beliefs", we will say that the system *knows* a statement if the system believes it and if that statement is actually true.

Truth. It is non-trivial to determine whether a statement is true. First of all, there exist certain statements that cannot have a truth value on principle. These include self-referring statements such as "This sentence is false" as well as sentences with invalid presuppositions such as "The Eiffel Tower is in Paris, the capital of Germany" or statements with void references such as "The King of France is blond". In the context of computer science, also undecidable statements such as the assertion that a program terminates fall into this category.

²adapted from [1] on "Belief"

³See [157] on "Belief" for an overview of these schools.

In this thesis, we will only deal with statements that do have a decidable truth value in the sense of computer science.

One problem for determining the truth value of a statement is the vagueness of our terms and notions. In this thesis, we will make the bold assumption that all terms and properties are crisp and well-defined. In some cases, the definition of the terms already entails the truth value of the sentence. For example, the statement “A bachelor is unmarried” is true simply by virtue of the definition of “bachelor”. These statements are called *analytic statements*. The remaining statements are called *synthetic*. The truth of synthetic statements may be highly subjective. Different people, different cultures and different religions consider different things true. Adopting the Correspondence Theory⁴, we will consider a statement true if and only if it corresponds to reality. We will not deal with statements that are exclusively subjective (such as “This book is interesting”).

Unfortunately, the concept of reality is assailable. Reality may well be just a product of our imagination. We would just “dream” there was a reality and we would have no way of figuring out that we are dreaming.⁵ As Descartes famously noted, the only thing a human mind can be sure of is its own existence. Taking a pragmatic attitude, I will call “reality” the system in which I perceive myself, whether it is a product of my imagination or not. In this view, *reality* is a unique system of things and states.

Epistemological Problems. Finding out whether a statement corresponds to reality is difficult. In science, the preferred approach is empirical, i.e., reproducible evidence is used to support a statement and reason is used to deduce new statements. This approach has a high predictive power and is thus highly useful, but it is restricted to statements about the physically observable world. Mental states, for example, typically fall outside this realm.⁶

Unfortunately, modern quantum physics has taught us that even in the observable world, the very observation of a state may alter that state. Thus, even in the observable world, there are areas in which we cannot prove statements by empirical evidence. Worse, the desire to give a justification for every statement entails the problem of *infinite regress*⁷: Each justification is again a statement and requires a justification. Thus, ultimately no statement has a circle-free finite chain of justifications. While well-aware of these problems, we will make the radical assumption that the truth of a statement can be verified. With this assumption, our reality is simply a set of true statements.

Understanding. Understanding means grasping the meaning of something [96]. This thesis will concentrate on the understanding of natural language texts. In this context, *understanding a text* will mean reconstructing the statements that the author of the text intended to express. When the text contains factual information, understanding the text will allow the machine to augment its knowledge about the world.

⁴See [157] on “Truth”

⁵This *dream argument* can be traced back to Plato and Aristotle. A related idea also resonates in Plato’s Allegory of the Cave. It appears prominently in the movie “The Matrix”.

⁶See [157] on “Qualia” for a discussion from a philosophical point of view.

⁷See [1] on “Regress argument”

Intelligence. It is unclear whether knowledge about the world actually makes a machine “intelligent”. Alan Turing proposed to call a computer intelligent if it can answer questions in such a way that a human cannot distinguish its answers from the answers given by a human. This so-called *Turing Test* has been criticized as anthropomorphic, because, with this definition, a machine could never be more intelligent than a human⁸. Common dictionaries [96] provide a more teleologic definition of intelligence: Intelligence is the ability to solve problems and to react to previously unknown situations. With this definition, whether a machine is intelligent or not will ultimately depend on how well it performs its duties under difficult circumstances.

1.3.2 Ontology

Ontology. While the previous section has established the concepts of *truth* and *reality*, this section will discuss how reality can be structured and described. The philosophical study concerned with this issue and with existence in general is called *Ontology*⁹. As in the area of Epistemology, there exist numerous puzzles and pitfalls in the field of Ontology. To overcome these difficulties, we will introduce a number of assumptions. They will allow us to structure the entities of this world into individuals, classes and relations.

Self-Reference. The field of Ontology is concerned with *existence* in general. This universality immediately entails a number of intrinsic problems. First, the study of Ontology itself exists. Hence it must be the subject of its own research. This phenomenon is called the problem of self-reflectivity [17]. Furthermore, the goal of Ontology is the definition of all concepts and terms. Consequently, the study of Ontology, unlike any other study, cannot rely on any predefined concepts or notions. In particular, this entails that it cannot define any term without being cyclic.¹⁰ To circumvent these philosophical problems, it has become common in computer science to make a number of implicit simplifying assumptions when representing knowledge. I have collected these implicit assumptions and coined the resulting knowledge representation model the *Computational Model of Ontology* [131, 132]. The reader is referred to this work [131] for a detailed description of the model with its philosophical problems and limitations. Here, we confine ourselves to an overview.

Entities. The basic notion of the Computational Model is the term *entity*. Any abstract or concrete thing, whether it exists or not, is an entity. The Computational Model makes a number of simplifying assumptions on entities: First, entities are assumed discernible. This view is by no means trivial, since there exist all kinds of variations, flows and transitions between entities. Drops of rain, for instance, fall down, join in a puddle and may be splattered by a passing car to form new drops [128]. Thus, our terminology of discrete entities

⁸See [1] on “Philosophy of artificial intelligence” for a discussion

⁹We follow Guarino’s distinction of “Ontology” (meaning the discipline) and “ontology” (meaning a certain conceptualization) [67].

¹⁰We discuss this problem in more detail in [49].

is just an approximation of reality; it is a grid through which we see only distinct things.

Furthermore, the Computational Model assumes the existence of atomic entities. In reality, however, entities are often non-atomic: Parts of a device get replaced by new parts, skin cells renew themselves, and countries can unite or break up. Notwithstanding these facts, the Computational Model assumes that entities can only be created and destroyed as wholes. This allows us to postulate a relation of identity, i.e., we assume that we can tell whether two entities are the same. The set of all entities that are in the scope of interest is called the *domain*.

Values and Words. Artificial identifiers such as numbers, date expressions, dimensioned quantities or strings are called *values*. Often (but not always), there exist infinitely many values of one type. Values are entities as well. Strings are certain types of values and hence words are also values. We say that a word *refers* to an entity, if the word is understood to mean the entity. For example, the word “*Einstein*” refers to the physicist Albert Einstein. We say that Albert Einstein is the *meaning* of the word “*Einstein*”. Inversely, the word “*Einstein*” is the *name* of the entity Albert Einstein. It is important to distinguish between words and their meanings, because words can have different meanings in different languages. Furthermore, the same entity can have different names (a phenomenon known as *synonymy*). Likewise, a word can have different meanings (a phenomenon known as *polysemy*). The process of determining which meaning is intended is called *disambiguation*.

Relations. The Computational Model assumes entities to have certain properties. For example, a leaf is green, a person can be blond and 2 is an even number. Since the definition of entities is so all-embracing, these properties are also entities. In the Computational Model, properties are seen as *unary*, *binary* and *n-ary relations* between entities.

Note that this choice does not explain why an entity has a property, what it means to have a property, or what the nature of properties is. It is just one arbitrary way of representing the phenomenon of properties. The formalization through relations is not unique. Although *hasColor(leaf, green)* would be the preferred representation [131], nothing prevents one from choosing *isGreen(leaf, yes)* or *is(leaf, green)* instead. A relation name together with an appropriate number of entities (such as *hasColor(leaf, green)*) is called a *statement*. Statements are entities as well. A statement that is true is called a *fact*. The arguments of a fact are called an *instance* of the relation. For example, *leaf/green* is an instance of the relation *hasColor*. The *domain* of a binary relation is the set of all entities that appear in first argument position. Analogously, the *range* of a binary relation is the set of all entities that appear in second argument position. For example, the range of *hasColor* is the set of colors. If, for a given first argument of a binary relation, there exists at most one second argument, the relation is called *functional*¹¹.

Classes. Entities with similar properties are grouped into sets called *classes*. This grouping, called *conceptualization*, is by no means unique and subject to

¹¹Functional relations are sometimes also called *right-unique*, *right-definite* or *many-to-one*.

ongoing discussion. The grouping of entities into classes can be motivated by psychological evidence for the existence of implicit conceptualizations in the human mind. This evidence, however, is disputed¹².

In the Computational Model, classes, as well as properties, are entities. Classes may or may not be part of the domain. If an entity is an element of a class, the entity is said to be an *instance* of the class. If a class is a subset of another class, the former is called a *subclass* of the latter. The set of instances of the *subClassOf* relation is called a *taxonomy*. The entities that are neither relations nor classes nor statements are called *individuals*¹³. Figure 1 shows the terminology. Each node in the graph is a class and the arrows denote the subclass relationship¹⁴.

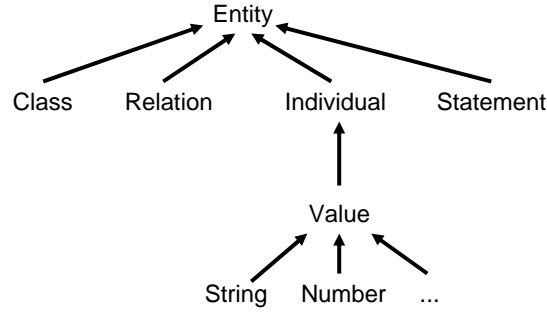


Figure 1: Ontological Terminology

ontology. An *ontology* (with lower case 'o') is the description of a domain, its classes and properties by means of a formal language. Some authors call only the description of classes and relations the *ontology*, and refer to the individuals and the relation instances as the *knowledge base* [91]. Since this distinction is not salient for the present work, we stick with the more general meaning of the word *ontology*.¹⁵ An ontology usually employs *symbols* that represent the entities. The goal of an ontology is to constrain its symbols in such a way that a large number of unintended interpretations of the symbols are ruled out [49]. It is common to call the symbols of the ontology "classes", "entities" and "relations", although, strictly speaking, these notions refer to the real-world objects and not to the symbols. A formal language used for an ontology is called a *knowledge representation model*. [116, 129] provide good overviews of this field.

¹²See Chapter 10 of [57] for a discussion.

¹³Sometimes, individuals are called *instances*. While every individual is an instance, a class may also be an instance, namely of the class *class*, see Section 2.1.3.

¹⁴Here, the problem of self-reflectivity appears: We are using the concepts of Ontology to define the concepts of Ontology.

¹⁵This is in line with recent blurring of the boundaries in [1, 5].

Chapter 2

Knowledge Representation

Knowledge representation is an old field in Artificial Intelligence. It has provided numerous models, from frames and KL-ONE to recent variants of description logics [116, 129]. Today, RDFS/OWL is the most common knowledge representation model. It has also been accepted and standardized by the World Wide Web Consortium¹. For our ontology YAGO, we will use a slightly modified version of this model. This variant is called the *YAGO model*. The following sections will first introduce the original RDFS/OWL model and then explain where and why the YAGO model deviates.

2.1 RDFS/OWL

The basis of RDFS/OWL is a knowledge representation model called *Resource Description Format* (RDF)[147]. Hence this section will first introduce RDF and then show how RDF is extended to the full RDFS/OWL model.

2.1.1 Symbols

RDF uses three types of symbols: URIs, literals and blank nodes.

URIs. A *URI* (Uniform Resource Identifier) can be seen as a generalized form of an Internet address. Any Internet address (such as `http://google.com/index.html`) is already a valid URI. Internet addresses identify resources that can actually be downloaded. URIs, in full generality, can also be non-downloadable. For example, `http://google.com/Elvis` is also a valid URI, even if it cannot be accessed on the Internet. Technically speaking, a URI is a string that follows a certain syntax specification[99]. From an ontological point of view, URIs are symbols that represent entities.² Different URIs can represent the same entity. For example, the following two URIs can both be intended to refer to the singer Elvis Presley:

```
http://google.com/Elvis
http://elvis-rulez.com/Elvis
```

¹<http://www.w3c.org>

²In RDF parlance, a URI *identifies* a resource, which in turn *refers to* an entity. In this thesis, we will restrict ourselves to talking of URIs and entities.

To simplify URIs, RDF allows defining *name spaces*. A name space is an abbreviation for the prefix of a URI. For example, in this thesis, we will use the name space *y* to abbreviate the URI prefix

`http://mpii.de/yago/resource/`

Then, *y:Elvis* stands for

`http://mpii.de/yago/resource/Elvis`

Literals. A *literal* is a string that represents a value. For example, the string "5" represents the value *five*. RDF groups values of the same type into *data types*. For example, all integers numbers make up the data type *Integer*. Technically speaking, a data type is a mapping from literals to values that specifies which literal represents which value. In addition to typed literals, RDF also knows *plain literals*, which are just strings with a language tag.

Blank nodes. In addition to URIs and literals, RDF also knows *blank nodes*. A blank node represents an object whose URI is not known or unimportant. Wherever a URI can be used, a blank node can be used instead.

2.1.2 Statements

To express that two entities stand in a binary relation, RDF uses a *statement*. A statement is a triple of three URIs: a *subject*, a *predicate* and an *object*. The statement says that the subject and the object stand in the relation given by the predicate. For example, to state that the entity *ElvisPresley* stands in the *hasWonPrize* relation with the entity *GrammyAward*, we write the statement

y:ElvisPresley y:hasWonPrize y:GrammyAward

Thus, a *statement* as used in RDF is simply a statement in the general sense (see Section 1.3.2), but restricted to binary relations and written with the relation between the entities. Note that in RDF, the relation, *hasWonPrize*, is also represented by a URI. The object of a statement can also be a literal. For example, to state that Elvis was born in the year 1935, we write:

y:ElvisPresley y:bornInYear 1935

2.1.3 Predefined Symbols

Beyond the data model, RDF also defines certain URIs. By convention, these use the name space *rdf*³. RDF is extended by the *Resource Description Framework Schema* (RDFS). This extension defines further URIs. These URIs conventionally use the name space *rdfs*⁴.

³*rdf* abbreviates the URI prefix <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

⁴*rdfs* abbreviates the URI prefix <http://www.w3.org/2000/01/rdf-schema#>

Predefined Classes. In RDFS, classes are represented by URIs – just like individuals and relations. The main predefined classes are

- *rdfs:Resource*. This class comprises all entities. Hence *resource* is a synonym for *entity* as defined in Section 1.3.2.
- *rdfs:Class*. This class comprises all classes. Each class (such as *y:singer*) will be an instance of *rdfs:Class*.
- *rdf:Property*⁵. This class comprises all binary relations. Thus, for RDFS, the terms *property* and *binary relation* are synonymous. Each relation (such as *y:hasWonPrize*) will be an instance of this class.

Predefined Relations. There exist also predefined relation symbols. One of them is the relation *rdf:type*, which says that the subject is an instance of the class given by the object. For example, to state that Elvis is a singer, we write

y:ElvisPresley *rdf:type* *y:singer*

The predefined relation *rdfs:subClassOf* allows us to state that every singer is a person:

y:singer *rdfs:subClassOf* *y:person*

RDFS also defines the relations *rdfs:domain* and *rdfs:range*. They allow restricting the domain and range of a relation, as in the following example:

y:bornOnYear *rdfs:domain* *y:person*
y:bornOnYear *rdfs:range* *y:integer*

Labels. Entities are referred to by *words*. Technically speaking, words are strings and thus they are entities as well. This makes it possible to express that a certain entity bears a certain name:

ElvisPresley *rdfs:label* “*Elvis*”

This allows us to deal with synonymy. The following line says that Elvis Presley is also known as “*Elvis Presley*”.

ElvisPresley *rdfs:label* “*Elvis Presley*”

Furthermore, we can express polysemy by stating that “*Elvis*” may also refer to the English songwriter Elvis Costello:

ElvisCostello *rdfs:label* “*Elvis*”

In a similar manner, RDFS defines additional relations such as *rdfs:subPropertyOf* and *rdfs:comment*. It also defines vocabulary for collections such as lists.

⁵For historical reasons, *Property* is defined in the name space *rdf* instead of *rdfs*.

2.1.4 Reification

RDF allows *reifying* statements, i.e., to make them objects of other statements. Assume for example that we wish to say that Bob believes the statement

y:Elvis y:livesOn y:Moon

To express this constellation in RDF, one needs to create a URI for the statement, say *y:elvisMoonStatement*. Next, one describes the statement *y:elvisMoonStatement* as follows:

<i>y:elvisMoonStatement</i>	<i>rdf:type</i>	<i>rdf:Statement</i>
<i>y:elvisMoonStatement</i>	<i>rdf:predicate</i>	<i>y:livesOn</i>
<i>y:elvisMoonStatement</i>	<i>rdf:subject</i>	<i>y:Elvis</i>
<i>y:elvisMoonStatement</i>	<i>rdf:object</i>	<i>y:Moon</i>

By this description, the statement did not become part of the knowledge base. For the system, Elvis does not live on the Moon. Instead, the statement became an entity that can now appear in statements:

y:Bob y:believes elvisMoonStatement

2.1.5 Extension to OWL

OWL. RDFS has been extended by additional vocabulary to become the *Web Ontology Language* (OWL). URIs defined by OWL conventionally use the name space *owl*⁶. Some new symbols are for example the following:

- *owl:sameAs* indicates that two URIs refer to the same entities.
- *owl:inverseOf* says that one relation is the inverse of another.
- *owl:disjointWith* means that the instances of two classes do not overlap.
- *owl:FunctionalProperty* is the class of all functional relations.

Restriction Classes. Furthermore, OWL allows *restriction classes*. A restriction class is a class that is characterized by certain limitations on its instances. For example, the class of all entities that are born in America is a restriction class. To state that all American singers are born in America, it suffices to make the class *y:AmericanSinger* a subclass of that restriction class. Thereby, *y:AmericanSinger* inherits the constraint that all of its instances must be born in America. Other types of restriction classes allow limiting the number of objects to which a relation may lead. For example, we can define the class of all entities that have less than 42 children.

Complexity of OWL. OWL also allows computing the intersection, union and complement of classes. Given that classes can be also restriction classes, operations on classes can be quite complex. One of the classical problems in knowledge representation is to determine whether a class can have instances at all, given the restrictions imposed on the class. This problem is known as the *satisfiability problem*. OWL is so complex that this problem is undecidable in OWL. This is why the language comes in three flavors:

⁶*owl* abbreviates the URI prefix <http://www.w3.org/2002/07/owl#>

1. OWL Lite is a subset of OWL that consists of only very few symbols and forbids certain types of statements. The expressive power of OWL Lite corresponds to the Description Logics $\mathcal{SHIF}^{(\mathcal{D})}$ [71]. Thereby, OWL Lite is the least expressive flavor, but reasoning in OWL Lite is still in NEXPTIME [71].
2. OWL DL is a superset of OWL Lite and imposes fewer restrictions. The expressive power of OWL DL corresponds to the Description Logics $\mathcal{SHOIN}^{(\mathcal{D})}$ [71]. This guarantees that the satisfaction problem on OWL DL is still decidable, although it is in EXPTIME [71].
3. OWL Full is the full set of OWL symbols without any constraints. It is the most expressive flavor of OWL, but it makes the satisfiability problem undecidable. OWL Full is the only flavor that allows reification.

2.1.6 Difficulties with the RDFS/OWL model

RDFS/OWL is a very powerful knowledge representation model, which allows a trading off expressiveness versus computational complexity. Still, it has two inconveniences for our purpose:

First, a knowledge gathering system will have to store with each fact where that fact was found. In the RDFS/OWL model, this would require reification. Reification, in turn, presumes OWL Full and OWL Full is undecidable. This problem seems to call for an alternative view of reification.

Second, the decidable flavors of OWL do not know the concept of acyclic transitive relations. This concept, however, is of utmost importance for an ontology, as all partial orders such as *subClassOf*, *partOf* and *locatedIn* are acyclic and transitive. This diagnosis seems to call for an extension of the original semantics.

2.2 The YAGO Model

The YAGO model [135] builds on RDFS. It extends RDFS by putting more emphasis on reification, by defining a more semantic data type hierarchy and by adding acyclic transitive relations. Most importantly, it defines a clean, decidable model-theoretic semantics. This section will first give an informal description of the YAGO model. It will then proceed with the formal model and the definition of semantics. Last, it will examine the relation of the YAGO model to RDFS and define a query language on the YAGO model.

2.2.1 Informal Description

The YAGO model adopts the complete RDFS knowledge representation model. It adopts most predefined relations. However, its symbols are not URIs, but simple, local identifiers. The YAGO model adds the class *atr*, which groups the acyclic transitive relations. It contains axioms that constrain relations such as *subClassOf* to acyclicity:

subclassOf *type* *atr*

Furthermore, the YAGO model equips each fact with a *fact identifier*. Fact identifiers are an integral part of the YAGO model. This allows talking about statements without explicitly reifying them. Take for example the fact

ElvisPresley *bornInYear* 1935

and suppose it has the fact identifier #1. Then the following line says that this fact was found in Wikipedia:

#1 *foundIn* *Wikipedia*

The YAGO model classifies entities into three groups:

- relations
- fact identifiers. These can be thought of as integer values.
- all other entities, called *common entities*. These include classes and all individuals, except for the fact identifiers.

In summary, a YAGO ontology is basically a function that maps fact identifiers to facts. More formally, a YAGO ontology can be described as a *reification graph*.

2.2.2 Reification Graphs

Reification Graphs. We generalize the notion of graphs as follows:

DEFINITION 1: [*Reification Graph*]

Given a set of nodes N , a set of edge identifiers I and a set of labels L , a *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 can connect not only 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 $\text{range}(G_{N,I,L})$.

YAGO Ontologies. Now, a YAGO ontology can be defined as follows:

DEFINITION 2: [*YAGO Ontology*]

A *YAGO ontology* y 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

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

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

$id_1:$ $arg1_1$ rel_1 $arg2_1$
 $id_2:$ $arg1_2$ rel_2 $arg2_2$
 ...

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

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

to mean

$id_1:$ $arg1_1$ rel_1 $arg2_1$
 $id_2:$ id_1 rel_2 $arg2_2$

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

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

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*

2.2.3 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:

<i>GrammyAward</i>	<i>prize</i>	<i>elvisGetsGrammy</i>
<i>Elvis</i>	<i>winner</i>	<i>elvisGetsGrammy</i>
1921	<i>year</i>	<i>elvisGetsGrammy</i>

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* *GrammyAward*

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

#2: #1 *inYear* 1967

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

(*Elvis hasWonPrize GrammyAward inYear 1967*)

or, with the assumption of left-associativity

Elvis hasWonPrize GrammyAward inYear 1967

More formally, the model handles n -ary relations as follows. Let r be an n -ary relation (e.g. *hasWonPrizeInYear*). Let a_1, \dots, a_n be the names of the arguments (e.g., *winner*, *prize*, *year*). Given a fact $r(x_1, \dots, x_n)$, the model allows some of the arguments to be unknown, i.e., $x_i = \text{null}$ for some i . For example, for *hasWonPrizeInYear*, the winner and the prize may be known, but the year might not, as in *hasWonPrizeInYear(Elvis, GrammyAward, null)*. The model assumes that (1) at least two arguments are known and (2) that the argument names a_1, \dots, a_n can be ordered in such a way that whenever one argument is known, all previous arguments are known as well. In the example, the sequence of arguments would be *winner*, *prize*, *year*. This assumes that the winner and the prize are always known. Let a_1, \dots, a_n be ordered in such a way. Then a fact $r(x_1, \dots, x_i, \text{null}, \dots, \text{null})$ can be expressed as

$f_1 :$	x_1	r	x_2
$f_2 :$	f_1	a_3	x_3
$f_3 :$	f_2	a_4	x_4
\dots			
$f_{i-1} :$	f_{i-2}	a_i	x_i

where f_1, \dots, f_{i-1} are arbitrary different fact identifiers.

This model assumes that the arguments of the relation can be ordered, so that whenever one argument is known, all previous arguments are known as well. This is a reasonable assumption for a number of relations, such as *hasWonPrizeInYear* (where the year is possibly unknown) or *actorPlaysRoleInMovie* (where the role might be unknown). In these cases, the model offers a convenient way to express facts. However, there may also be cases where it is difficult to order the arguments a priori, for example for a relation that has a large number of arguments. In these cases, the model can still accommodate event entities in the spirit of RDFS and OWL.

2.2.4 Data Types

YAGO Data Types. The RDFS/OWL model uses the data types defined by XML Schema [149]. 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 disjoint data types *float* and *double*. This is why the YAGO model comes with its own data types (see Figure 2), which follow the SUMO ontology [102].

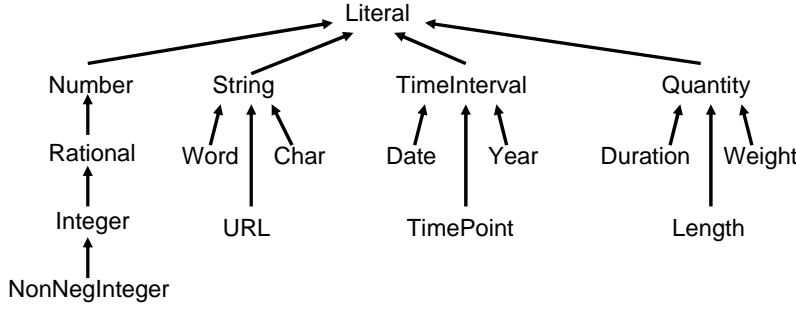


Figure 2: The YAGO literal classes

YAGO sees, e.g., *integer* as a subclass of *rational*, because each integer number is a rational number. *timeIntervals* are specific periods of time, such as the year 2007 or the 8th of January 1935.

Quantities. 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 blank nodes. This entity is connected by an *rdf:value* edge to the numerical value and by a *unit* edge to the unit of measurement, for example 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.

2.2.5 Semantics

Prerequisites. 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 2.

For the rest of this section, 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 $\mathcal{I} = \text{domain}(y)$. To define the semantics of a YAGO ontology, we consider the set of all possible facts $\mathcal{F} = (\mathcal{I} \cup \mathcal{C} \cup \mathcal{R}) \times \mathcal{R} \times (\mathcal{I} \cup \mathcal{C} \cup \mathcal{R})$.

Rewrite System. 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 $\{f_1, \dots, f_n\} \hookrightarrow f$ to say that

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

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

$$\begin{aligned} \emptyset &\hookrightarrow (\text{domain}, \text{range}, \text{class}) \\ \emptyset &\hookrightarrow (\text{domain}, \text{domain}, \text{relation}) \\ \emptyset &\hookrightarrow (\text{range}, \text{domain}, \text{relation}) \\ \emptyset &\hookrightarrow (\text{range}, \text{range}, \text{class}) \\ \emptyset &\hookrightarrow (\text{subClassOf}, \text{type}, \text{atr}) \\ \emptyset &\hookrightarrow (\text{subClassOf}, \text{domain}, \text{class}) \\ \emptyset &\hookrightarrow (\text{subClassOf}, \text{range}, \text{class}) \\ \emptyset &\hookrightarrow (\text{type}, \text{range}, \text{class}) \\ \emptyset &\hookrightarrow (\text{subRelationOf}, \text{type}, \text{atr}) \\ \emptyset &\hookrightarrow (\text{subRelationOf}, \text{domain}, \text{relation}) \\ \emptyset &\hookrightarrow (\text{subRelationOf}, \text{range}, \text{relation}) \end{aligned}$$

The first rule, for example, 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 2.

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) $\{(x, \text{type}, c_1), (c_1, \text{subClassOf}, c_2)\} \hookrightarrow (x, \text{type}, c_2)$

Properties of the Rewrite System.

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 B.1. Given a YAGO ontology y , the rules of \rightarrow can be applied to its set of facts, $\text{range}(y)$. We call the largest set that can be produced by applying the rules of \rightarrow the *set of derivable facts* of y , $D(y)$. This allows us to define when two YAGO ontologies are equivalent:

DEFINITION 3: [*Equivalence of YAGO ontologies*]

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

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

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 define the deductive closure more formally as follows:

DEFINITION 4: [*Deductive Closure*]

The *deductive closure* of a YAGO ontology y is the YAGO ontology y^* given by

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

Structures. To define the semantics of YAGO, we need the notions of *structure* and *interpretation*.

DEFINITION 5: [*Structure*]

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.

DEFINITION 6: [*Interpretation*]

An *interpretation* with respect to a structure $\langle \mathcal{U}, \mathcal{D}, \mathcal{E} \rangle$ is the relation Ψ with

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

We say that a fact (e_1, r, e_2) is *true* in a structure, if it is contained in the interpretation. Interpretations that make all facts of the ontology true are called *models*:

DEFINITION 7: [*Model*]

A *model* of a YAGO ontology y is a structure such that

1. all facts of y^* are true in the structure
2. if $\Psi(x, \text{type}, \text{string})$ for some x , then $\mathcal{D}(x) = x$
3. if $\Psi(r, \text{type}, \text{atr})$ for some r , then there exists no x such that $\Psi(x, r, x)$

Consistency. The notion of models allows us to define finally the consistence of a YAGO ontology:

DEFINITION 8: [*Consistency*]

A YAGO ontology y is called *consistent* if there exists a model for it.

Obviously, a YAGO ontology is consistent iff

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

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.

Bases. Now we turn to considering ways of minimizing a YAGO ontology.

DEFINITION 9: [*Bases*]

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.

Canonical bases have a convenient property in YAGO:

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.2. 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.

2.2.6 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 such as *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.

2.2.7 Relation to RDFS/OWL

Vocabulary. The YAGO model is an extension of RDFS. It maintains the semantics of the RDFS relations *domain*, *range* and *type*. It also maintains the RDFS relations *subClassOf* and *subRelationOf* (*subPropertyOf* in RDFS). However, the YAGO model adds acyclicity to these relations. RDFS, in contrast, does not know the concept of an acyclic relation. This entails that the relation *atr* can be defined and used, but that RDFS would not know its semantics. Another difference to RDFS, discussed in Section 2.2.4, is the use of semantic data types in YAGO (such as rational numbers). As a small syntactic deviation, the YAGO model uses the relation symbol *means* instead of *rdfs:label*. Thus, the fact that *ElvisPresley* is called "*Elvis*" becomes

"Elvis" *means* *ElvisPresley*

Reification. Just as RDFS, the YAGO model uses statements identifiers to express statements about statements. While RDFS requires explicit reification, the YAGO model treats fact identifiers as an integral part of the model. On the semantic level, the difference is that the YAGO model can only talk about facts that are part of the ontology. In RDFS, arbitrary facts can be used as arguments, even ones that are false in the model.

Semantics. YAGO uses fact identifiers, but it does not have built-in relations to make logical assertions about facts (e.g., it does not allow 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[106]. The extended description logic *SR_QIQ* [70], which has been adopted as the logical basis of OWL 1.1, allows expressing irreflexivity and transitivity. This allows defining acyclic transitivity, even though *subClassOf* 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.

2.2.8 Query Language

To demonstrate the use of YAGO, we present a query language for reification graphs. The query language can express conjunctive queries in the sense of SQL. It is similar to SPARQL [148], but it allows querying reified facts, which is not straightforward in SPARQL.

Queries. Let us start the discussion of our language by defining a *query*. In our language, a query is a reification graph:

DEFINITION 10: [*Query*]

A *query* for a reification graph $G_{N,I,R}$ over a set of variables V , $V \cap (N \cup I \cup R) = \emptyset$, is a reification graph over the set of nodes $N \cup V$, the set of identifiers $I \cup V$ and the set of labels $R \cup V$.

In the following, we denote elements of V by symbols that carry a question mark (such as $?x$). With this convention, a query for the question “*When did Elvis win the Grammy Award?*” looks as follows:

?i1:	Elvis	hasWonPrize	GrammyAward
?i2:	?i1	inYear	?x

Our syntax simplifications from Section 2.2.2 can be transferred to patterns: Each implicit fact identifier becomes a fresh variable. As a result, we could also formulate our query as

Elvis hasWonPrize GrammyAward inYear ?x

For a query, we also define a *result* in the ontology:

DEFINITION 11: [*Variable Binding, Result*]

A *variable binding* for a query Q with variables V on a reification graph $G_{N,I,L}$ is a substitution $\sigma : V \rightarrow N \cup I \cup L$, such that $\sigma(Q) \subset G$. $\sigma(Q)$ is called a *result* of the query.

A variable binding for a query maps the variables to entities so that the query becomes a subgraph of the ontology. In our example, the variable binding would map $?x$ to 1967 and $?i1$, $?i2$ to some fact identifiers. Thus, a result of this query could be:

42:	Elvis	hasWonPrize	GrammyAward
43:	42	inYear	1967

This graph is a subgraph of the ontology. In shorthand notation, it becomes

(Elvis hasWonPrize GrammyAward) inYear 1967

or, with left-associativity,

Elvis hasWonPrize GrammyAward inYear 1967

Queries with Virtual Relations. Usually, each entity that appears in the query also has to appear in the ontology. If that is not the case, there is no possible variable binding. 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 *virtual relations* (such as *after*), which are not part of the result, but are evaluated on the result as filters:

DEFINITION 12: [*Virtual Relation*]

A *virtual relation* is a decidable function that maps a pair of literals to either 0 or 1.

This allows us to generalize queries to *queries with virtual relations*:

DEFINITION 13: [*Query with Virtual Relations*]

A *query* for a reification graph $G_{N,I,R}$ over a set of virtual relations Z , a set of literals L and a set of variables V , $V \cap (N \cup I \cup R \cup L) = \emptyset$, 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 $R \cup V \cup Z$.

For example, the following is a query over the set of literals $\{1930\}$ and the set of virtual relations $\{after\}$:

?i1:	?x	type	singer
?i2:	?x	bornInYear	?y
?i3:	?y	after	1930

We extend the notion of *results* to queries with virtual relations as follows:

DEFINITION 14: [*Variable Binding, Result with Virtual Relations*]

A *variable binding* for a query Q over the set of virtual relations Z with variables V on a reification graph $G_{N,I,L}$ is a variable binding σ for the query $Q \setminus \{(i, (a_1, r, a_2)) | r \in Z\}$, such that

$$\forall (i, (a_1, r, a_2)) \in Q, r \in Z : r(\sigma(a_1), \sigma(a_2)) = 1.$$

$\sigma(Q)$ is called a *result* of the query.

In the example, a variable binding would have to bind ?x and ?y in such a way that $after(?y, 1930) = 1$. The variable ?i3 is left unbound. Thus, a result for this query could be for example

#1:	Elvis	type	singer
#2:	Elvis	bornInYear	1935
?i3:	1935	after	1930

We will see applications of the query language in Section 3.3.3.

2.3 Summary

This section has discussed different models for knowledge representation. The most popular model at the time is the RDFS/OWL model. It expresses statements by triples of two entities and one relation. We have seen that this model brings two inconveniences for our purpose: First, reification is only possible in the undecidable flavor of OWL. Second, acyclic transitive relations are not supported in RDFS.

This is where the YAGO model takes over. It extends RDFS by a new reification mechanism, by new data types and by acyclic transitive relations. The formal basis of the model are reification graphs. We showed that reification graphs allow expressing n -ary relations in an elegant way. 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. Last, we also introduced a query language that is expressive enough to handle queries over reification graphs.

Chapter 3

YAGO

The goal of this thesis is to describe a system that can grow knowledge automatically. This system consists of three components: A core ontology, an information extraction tool and an integration mechanism. This chapter introduces the first of the three components, the ontology YAGO¹[136, 135]. The chapter also presents the methodology and techniques used to build YAGO. The first section introduces the problem and related work. The second section describes how YAGO is constructed and the third section presents an evaluation.

3.1 Overview

3.1.1 Problem Statement

This chapter discusses the task of creating a general-purpose ontology (see Section 1.3.2). Our aim is to construct a knowledge base that contains

1. Individuals. We are interested in all kinds of individuals, including, for example, cities, people, organizations, companies, movies, and also abstract things like historic events or philosophical theories.
2. Classes. Each individual shall be an instance of at least one class. The classes shall be arranged in a hierarchic taxonomy.
3. Relations. We are interested in different relations, such as *bornOnDate*, *locatedIn*, *hasInflation* and many others.
4. Facts. Our ontology shall contain facts about the entities. We are interested in as much factual information as possible.

For example, our ontology shall contain facts of the following form

<i>ElvisPresely</i>	<i>type</i>	<i>singer</i>
<i>ElvisPresely</i>	<i>bornOnDate</i>	<i>1935-01-08</i>
<i>ElvisPresely</i>	<i>bornIn</i>	<i>Tupelo</i>
<i>Tupelo</i>	<i>locatedIn</i>	<i>Mississippi(state)</i>
<i>Mississippi(state)</i>	<i>locatedIn</i>	<i>USA</i>
...		

¹Yet Another Great Ontology

In general, we aim for a domain-independent ontology. Our ontology shall not be restricted to a certain topic, but it shall contain general knowledge about the world. Furthermore, we would like to minimize human effort in the construction. That is, we would like the ontology to be constructed automatically. Automatic gathering of knowledge often brings along a decrease in accuracy. For example, automatic techniques for ontology construction often involve information extraction from natural language text, which may introduce false information. In our case, however, we aim at an ontology of high accuracy. That means, the ontology shall contain statements that are actually true with high probability (see Section 1.3.2).

3.1.2 Related Work

Numerous approaches have been proposed to build a general purpose ontology.

Text-Based Approaches. One class of approaches focuses on extracting knowledge structures automatically from text corpora. They use information extraction technologies that include pattern matching, natural-language parsing, and statistical learning [2, 124, 105, 45]. Two important projects along these lines are KnowItAll [56] and TextRunner [10]. KnowItAll aimed at extracting instances of a given set of unary or binary relations on a very large scale. 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* [55]. 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 [11]. Except for [11], all of these systems extract facts in a non-canonical form. This means that different identifiers are used for the same entity and there exist no clearly defined relations. A sample fact (taken from the output of [10]) is

0.0% are under the age of 18

As a result, no explicit (logic-based) knowledge representation model is available. Furthermore, the quality of text-based systems is still significantly below that of a hand-crafted knowledge base. If facts have an associated confidence measure, it is often just a real valued score, which is hard to interpret. Thus, text-based approaches are still much more suitable for high coverage and less attractive for applications that need consistent ontologies (such as 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., [53, 77, 54]). Notwithstanding the benefits of these approaches, the inherent noise and lack of explicit quality control for social tagging usually lead to poor precision.

Man-Made Ontologies. Because of the quality bottleneck, the most successful and widely employed ontologies are still man-made. These include WordNet [59], Cyc or OpenCyc [95], SUMO [102], and especially domain-specific ontologies and taxonomies such as UMLS² or the GeneOntology³. These knowledge

²<http://umlsinfo.nlm.nih.gov>

³<http://www.geneontology.org>

sources have the advantage of satisfying the highest quality expectations, because 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.

Community-Based Approaches. Lately, a new approach has entered the scene: community-based ontology building. Inspired by Wikipedia, the Freebase project⁴ aims to construct an ontology by inviting volunteers to contribute facts. The usefulness of this approach will depend on the acceptance of the project by the community. Furthermore, effective ways of enforcing uniformity across the ontology need to be found, as different contributors may prefer different ways of modeling reality. To facilitate the startup, the Freebase project has incorporated the YAGO ontology into its knowledge base. The Semantic Wikipedia project [87] 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.

Semi-Structured Approaches. Finally, a recently emerging approach is to automatically derive explicit facts from the semi-structured part of Wikipedia. One of the first projects in this direction was DBpedia [8]. DBpedia extracts facts from the infoboxes of Wikipedia. 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. In contrast to YAGO, DBpedia does not use defined relations with ranges and domains. Rather, it uses the words from the infoboxes as relation names. This way, DBpedia can extract a wealth of facts from the infoboxes. As a drawback, the same relationship may appear with different names (e.g. *length*, *length-in-km*, *length-km*). Furthermore, DBpedia has not been subjected to an evaluation so far. Thus, the consistency and accuracy of DBpedia are unknown. DBpedia uses YAGO as a taxonomic backbone to connect the facts to a coherent whole.

Ponzetto et al. [107] use rich heuristics to derive a taxonomy from Wikipedia categories and links between them. Isolde [145] extracts class candidates from a specific domain corpus. It exploits Web sources such as Wikipedia and Wiktionary to derive additional knowledge about these candidates. Both of these approaches aim at a taxonomy rather than a full-fledged ontology.

The PORE algorithm [143] combines extraction from Wikipedia infoboxes with extraction from the natural language part of Wikipedia. Zirn [163] show how the category names in Wikipedia can be split into classes and individuals. Both approaches provide interesting insights into the wealth of information hidden in Wikipedia, but they do not aim at constructing a full general ontology.

KYLIN [150] starts out with extraction techniques on infoboxes, similar to those of DBpedia, but then uses powerful learning techniques to automatically fill in missing values in incomplete infoboxes. The accuracy of the extraction is remarkable. Its goal, however, is filling infoboxes rather than constructing

⁴<http://www.freebase.com>

an ontological knowledge base. At this point, the KYLIN Ontology Generator (KOG) [151] takes over. Its goal is to use KYLIN to build a comprehensive knowledge base. KOG uses and refines the taxonomy of YAGO. The result, the KOG ontology, is not publicly available.

Linking Open Data. There is also a meta-approach to ontology construction: The Linking Open Data Project [18], launched by the W3C, aims to interlink existing ontologies. It encourages people to make RDFS data sets available online as Web services. On top of these Web services, it establishes links between equivalent concepts in different data sets. DBpedia [8] has pioneered this area. By being linked to DBpedia, YAGO also becomes part of this knowledge network.

3.1.3 Contribution

This section presents YAGO [135, 136], an ontology that combines high coverage with high quality. Its core is extracted automatically from one of the most comprehensive lexicons available today, Wikipedia. The facts from Wikipedia are combined with the taxonomic structure of WordNet [59].

The key contributions of YAGO are the following:

1. **Information Extraction from Wikipedia.** Our approach builds on the *infoboxes* and *category pages* in Wikipedia. As shown in [8], infoboxes can be exploited to yield huge numbers of facts. Category pages are lists of articles that belong to a specific category (e.g., Elvis is in the category of American rock singers). As shown in [85], category pages can be used to establish type information (e.g. *type(Elvis, rockSinger)*) and also other facts (e.g. *nationality(Elvis, American)*). We present techniques that harvest the infoboxes and category pages in a more systematic manner, yielding millions of highly accurate facts.
2. **Combination with WordNet.** In an ontology, classes have to be arranged in a taxonomy for useful type information. 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 classes. But the Wikipedia concepts have no obvious counterparts in WordNet.

We present techniques that link the two sources with high accuracy. 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.

3. **Quality Control.** We explain how we can enforce the high accuracy of our extraction 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 that are entailed).

These techniques yield a rich ontology of entities and relations, currently containing more than 2 million entities and more than 19 million facts. An extensive evaluation study proves that the accuracy of YAGO stands at 95%.

3.1.4 Sources for YAGO

YAGO bases on two sources: WordNet and Wikipedia. This subsection briefly presents the two resources.

3.1.4.1 WordNet

WordNet is a semantic lexicon for the English language developed at the Cognitive Science Laboratory of Princeton University[59]. WordNet distinguishes between words and the meanings of the words. A set of words that share one meaning is called a *synset*⁵. Thus, from an ontological point of view, a synset represents an entity. The elements of a synset are the names of the entity. 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 such as verbs and adjectives, but we consider only nouns in this thesis.) WordNet provides relations between synsets such as *subClassOf* (called hypernymy in WordNet) and *partOf* (called meronymy in WordNet). Conceptually, the hypernymy relation in WordNet spans a directed acyclic graph (DAG) with a single root node called *entity*.

3.1.4.2 Wikipedia

Wikipedia is a multilingual, Web-based encyclopedia. It is written collaboratively by volunteers and is available for free under the terms of the GNU Free Documentation License⁶. As of September 2008, the English Wikipedia contained more than 2 million articles. Each Wikipedia article is a single Web page and usually describes a single topic or entity. Figure 3 shows the article about Elvis Presley.

As an online encyclopedia, Wikipedia has several characteristics: First, each article is highly interlinked to other articles. In the example, a click on the word *singer* leads to the Wikipedia article about singers. Each Wikipedia article has an unstructured, natural language part (on the left hand side in Figure 3). In addition to that, some Wikipedia articles also have an infobox (pictured on the right hand side). 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. In our example, the infobox for musical artists is used. Each row in the infobox contains an attribute and a value. For example, our infobox contains the attribute *Birth*

⁵There exist synsets that contain exactly the same words but represent different meanings. For example, there are two synsets, which each contain only the word “*abstraction*”, but which are considered different nevertheless.

⁶See <http://www.gnu.org/copyleft/fdl.html>.

name with the value *Elvis Aaron Presley*. An infobox attribute may also have multiple values, as exemplified by the *Occupations* attribute. The majority of Wikipedia articles have been manually assigned to one or multiple *categories*. Our sample article is in the categories *American rock singers*, *1935 births*, and 34 more. Figure 3 shows an excerpt at the bottom.

Elvis Presley

From Wikipedia, the free encyclopedia


Elvis Aaron Presley ([January 8, 1935](#) – [August 16, 1977](#)), middle name sometimes written Aron) was an [American singer](#), [musician](#) and [actor](#). A cultural icon, he is commonly referred to by his first name, and as the "The King of Rock 'n' Roll" or "The King".

etc.

Categories: [1935 births](#) | [1977 deaths](#) | [American rock singers](#) | [Rock and roll](#) | [People with diabetes](#)

etc.

Elvis Presley



Background Information

Birth name: Elvis Aaron Presley
 Born: January 8, 1935
 Died: August 16, 1977
 Genre(s): [Rock and roll](#)
 [Country rock](#)
 Occupations: [singer](#), [actor](#)
 Website: <http://elvis.com>
 etc.

Figure 3: A Wikipedia Article

Wikipedia is rendered as HTML pages, but is written in a special markup language, the *Wiki markup language*. Figure 4 shows an excerpt of the article on Elvis Presley in this language. For our information extraction, we used the XML dump of Wikipedia as of September 2008. It is approximately 18 Gigabytes large and stores the articles in the Wiki markup language.

```

{{Infobox musical artist
| Name = Elvis Presley
| Img = Elvis Presley 1970.jpg
| Birth name = Elvis Aaron Presley
| Born = {{birth date|1935|1|8|}}
| Died = {{Death date|1977|08|16|}}
| Genre = [[Rock and roll]], [[Country rock]]
| Occupation = [[singer]], [[actor]]
| URL = [http://elvis.com]
etc.
}}

'''Elvis Aaron Presley'''([[January 8]], [[1935]] -- [[August
16]], [[1977]]), middle name sometimes written '''Aron''',
was an [[United States|American]] [[singer]], [[musician]] and
[[actor]]. He is considered a cultural icon, recognized simply
by his first name. He is also referred to as the ''King of Rock
'n' Roll'', or as ''The King''.
etc.

[[Category:1935 births]]
[[Category:1977 deaths]]
[[Category:American rock singers]]
[[Category:Rock and roll]]
[[Category:People with diabetes]]
etc.

```

Figure 4: The Wikipedia Markup Language

3.2 Construction of YAGO

The construction of the YAGO ontology takes place in two stages: First, different information extraction methods 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 examine these two steps in detail and afterwards see how YAGO is stored.

3.2.1 Information Extraction

Since Wikipedia knows far more individuals than WordNet, the individuals for YAGO are taken from Wikipedia. Each Wikipedia page title is a candidate to become an individual in YAGO. For example, the page title “*Albert Einstein*” is a candidate to become the individual *AlbertEinstein* in our ontology. The page titles in Wikipedia are unique. Our algorithm parses the XML dump of Wikipedia and applies 4 different types of extraction techniques to the articles.

3.2.1.1 Infobox Harvesting

The Attribute Map. A Wikipedia article may contain an infobox (see Figure 3). We harvest the infoboxes by help of an *attribute map*:

DEFINITION 15: [*Attribute Map*, *Target Relation*]

An *attribute map* is a function that maps infobox attributes to relations. The relation of an attribute is called the *target relation* of the attribute. An entry of the attribute map can be marked as *inverse*, *manifold* or *indirect*.

We will now explain the entries of this map and the markers for the entries. Each entry of the map maps one infobox attribute (such as *Born*) to a corresponding YAGO relation (such as *bornOnDate*). For YAGO, we have identified 170 highly frequent attributes. For each of these attributes, we have manually designed a target relation with domain and range. One attribute can only map to one single target relation. However, different attributes may map to the same relation. For example, both *Born* and *Birthday* map to the relation *birthDate*. There are cases in which an attribute has to be mapped 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 introducing a relation *hasOfficialName*, which has the name as its second argument, we would like to map the attribute to the *means* relation, which has the name as its first argument. For this purpose, the entry in the attribute map for *official name* can be marked as *inverse*.

Some attributes may have multiple values. For example, a person may have multiple children. In this case, the entry in the attribute map may be marked as *manifold*. 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 *hasGDP* gdp *during* year

Entries for this type of attributes are marked as *indirect* in the attribute map. 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). Appendix A.1 lists the whole attribute map.

Algorithm. The algorithm for harvesting infoboxes is simple: It parses all Wikipedia articles. Once it finds an infobox, it walks through all of its attributes. It looks up each attribute in the attribute map. If an attribute is listed in the map, 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 parser from [134] to parse literals of different types (see Section 4.2.1). 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.

3.2.1.2 Type Extraction

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 (such as *1879 births*) and again others indicate merely thematic vicinity (such as *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. For example, a category name such as *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 [134] to identify and stem plural words (see Section 4.2.1). This gives us a (possibly empty) set of conceptual categories for each Wikipedia page. Conveniently, articles that do not describe individuals (such as 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. For example, the category *People by occupation* is a sub-category of *Business*. If categories were interpreted as classes, this would mean that every person with an occupation is a business – which is absurd. Hence, the hierarchy is of little use from an ontological point of view. Therefore, 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 we ensure 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 Algorithm 1.

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*).

ALGORITHM 1: wiki2wordnet

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  $s_1, \dots, s_n$  for  $head$ 
8      (ordered by their frequency for  $head$ )
9      RETURN  $s_1$ 
10 FAIL

```

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, \dots, 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*. It would be possible to introduce another intermediate class, so that *American_people_in_Japan* becomes a subclass of *American_person*, which is again a subclass of *person/human*. Since there are only very few cases in which a category name has both a pre-modifier and a post-modifier, we waived this possibility.

Exceptions. There were only around two dozen prominent cases in which the disambiguation of the Wikipedia category names failed. For example, all categories with the head compound “*capital*” in Wikipedia mean the “*capital city*”, but the most frequent sense in WordNet is “*financial asset*”. We corrected these cases manually. In summary, we obtain a complete hierarchy of classes, where the upper classes stem from WordNet and the leaves come from Wikipedia.

3.2.1.3 Word Level Techniques

Exploiting WordNet Synsets. WordNet also yields information on the meaning of words. For example, the word “*metropolis*” belongs to the synset

city, implying that one meaning of “*metropolis*” is *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 a 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*, *AlbertEinstein*)).

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 *AlbertEinstein* is a person, so we introduce the facts (“*Einstein*”, *familyNameOf*, *AlbertEinstein*) and (“*Albert*”, *givenNameOf*, *AlbertEinstein*). Both are subrelations of *means*, so that the family name “*Einstein*”, for example, also means *AlbertEinstein*.

3.2.1.4 Category Harvesting

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 harvest category names by a *category map*:

DEFINITION 16: [*Category Map*, *Target Relation*]

A *category map* is a function that maps regular expressions on category names to relations. The relation of a regular expression is called the *target relation* of the regular expression.

For example, the category map can map the regular expression “*Mountains|Rivers in (.*)*” to the target relation *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, for example, the *Rhine* is in the category *Rivers in Germany*, then we add the fact (*Rhine*, *locatedIn*, *Germany*). Table 1 shows our category map.

Since all candidate facts will be type checked, we can be generous with our heuristics. For example, the last two map entries 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 3.2.2.2).

Table 1: Category Map

Regular Expression	Target Relation
$([0-9]\{3,4\})$ births	<i>bornOnDate</i>
$([0-9]\{3,4\})$ deaths	<i>diedOnDate</i>
$([0-9]\{3,4\})$ establishments	<i>establishedOnDate</i>
. * established in $([0-9]\{3,4\})$	<i>establishedOnDate</i>
$([0-9]\{3,4\})$ books novels	<i>writtenOnDate</i>
Mountains Rivers <i>etc.</i> in (.)	<i>locatedIn</i>
Presidents Governors of (.)	<i>politicianOf</i>
(.) winners laureates	<i>hasWonPrize</i>
$[A-Za-z]^+ (.)$ winners	<i>hasWonPrize</i>

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*

3.2.2 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.

3.2.2.1 Canonicalization

Redirect Resolution. Our infobox algorithm delivers 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 extraction algorithms 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 harvesting has determined a birth date of *1935* and the infobox harvesting has determined *1935-01-08*, then only the fact with *1935-01-08* is kept.

3.2.2.2 Type Checking

Reductive Type Checking. A candidate fact may contain an entity for which the extraction algorithm 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 relation, this fact is eliminated (similarly for the second argument and the range). This type constraint also applies to literals, but the extraction algorithms already make sure that literals have the correct data type.

Inductive Type Checking. Type constraints cannot only be used 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 person 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.

Type Coherence Checking. In some cases, the classification yields wrong results. For example, Abraham Lincoln is an instance of 13 classes. 12 of them are subclasses of the class *person* (such as *lawyer* and *president*). The 13th class is the class *cabinet*, which has been extracted by mistake. To clean these erroneous classifications, we use *Type Coherence Checking*, a technique we developed in [49]: At the top level, the class hierarchy of YAGO is partitioned into different branches. These include locations, artifacts, people, other physical entities, and abstract entities. If a YAGO individual is an instance in multiple branches, a voting procedure is used to determine the branch that most *type* facts lead to (breaking ties arbitrarily). These *type* statements are kept and all others are purged. This decreases the number of *type* statements by roughly 10% . In return, each individual belongs to exactly one branch and potential errors in the YAGO taxonomy are removed.

3.2.3 Storage and Export Formats

3.2.3.1 Storage Format

Descriptions. Due to its generality, the YAGO ontology can store meta-relations uniformly together with usual relations. For example, we store for each individual the URL of the corresponding Wikipedia page. This allows 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, for example, 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 3.3).

3.2.3.2 Export Formats

We provide conversion programs to convert the ontology to different output formats.

XML. YAGO is available as a simple XML version. This version of YAGO mirrors the folder structure of the original files in one huge XML file, together with a DTD file.

Database. YAGO can be loaded easily into a database. One way to store YAGO in a database would be to use one table per relation. This could potentially speed up queries in which the relation is known. However, this representation would complicate queries in which the relation is unknown. It would also complicate queries that aim to discover a path between two entities in the ontology. Hence, our table has the simple schema

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

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

RDFS. Since the YAGO model is syntactically compatible with RDFS, we also provide an RDFS version of YAGO. We take care to map the YAGO data types to RDFS data types wherever possible. Furthermore, we map the *is-Called/inLanguage* facts to appropriate plain literals with language tags, as supported by RDFS. The *means* relation is mapped to *rdfs:label*. YAGO makes heavy use of reification. It uses reification both to store witnesses and to represent *n*-ary relations. Fortunately, the XML syntax of RDFS allows a short cut for this process: If a fact of the ontology is equipped with a triple identifier, that fact is automatically reified. We define XML entities to further cut down the verbosity of XML. In the end, the fact

Elvis *hasWonPrize* *GrammyAward* *inYear* *1967*

is output as

```

<rdf:Description rdf:about="&y;Elvis">
  <yago:hasWonPrize rdf:ID="f1" rdf:resource="&y;GrammyAward">
</rdf:Description>
<rdf:Description rdf:about="&y;f1">
  <y:inYear rdf:ID="f2" rdf:datatype="&d;decimal">1967</yago:inYear>
</rdf:Description>

```

Web service. YAGO is also available as a Web service. The Zitgist LLC. company⁷ kindly proposed to host the YAGO service on its servers⁸. This service is based on the RDFS version of YAGO. It allows Semantic Web applications to access YAGO online in a machine-readable format.

DBpedia Export. YAGO forms the taxonomic backbone of the DBpedia ontology [8]. We provide template code for exporting the class hierarchy of YAGO to DBpedia.

DBpedia Interlinks. The Linking Open Data project [18] encourages people to publish existing ontologies as Web services. If two symbols of different ontologies refer to the same entity, these two symbols shall be interlinked and marked as equivalent. The DBpedia project [8] has pioneered this area and has connected the DBpedia ontology to dozens of other ontologies. By providing links between equivalent symbols in YAGO and DBpedia, YAGO has become part of this knowledge network. The DBpedia team kindly hosts the equivalence links as part of their Web service.

3.2.4 Query Engine

Queries. We implemented a simple query engine along the lines of [81] on top of the database version of YAGO. It can solve queries of the form described in Section 2.2.8. 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

⁷<http://www.zitgist.com/>

⁸ Accessible at <http://umbel.zitgist.com/yago.php>

Results. A result of 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.

Implementation. In the deductive closure, an individual is an instance of all super-classes of its class. Since many queries ask for the class an individual belongs to, we pre-computed the deductive closure of the *type/subclassOf*-axiom, so that each individual is connected by a *type* fact to all of its super-classes. This implementation leaves much room for improvement, especially concerning efficiency. For example, it takes several seconds to return 10 results to the query “*Who was born after Elvis?*”. Queries with more joins can take even longer. In this thesis, we use the engine only to showcase the contents of YAGO.

3.3 Evaluation and Demonstration

3.3.1 Precision

Evaluation Setup. 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 a snippet of the corresponding Wikipedia page. Thus, our evaluation compared YAGO against the ground truth of Wikipedia (i.e., it does not deal with the problem of Wikipedia containing 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 heuristic. 13 judges participated in the evaluation and evaluated a total number of 5200 facts.

Results. We report the precision of the most precise and least precise extraction techniques in Table 2. To be sure that our findings are significant, we computed the Wilson interval [23] for $\alpha = 5\%$.⁹ A confidence interval of 0% means that the facts produced by the heuristic have been evaluated exhaustively. A complete list of all heuristics with their precisions can be found in Appendix A.3.

Table 2: Precision of YAGO’s heuristics

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

Discussion. 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. *establishedInYear* is a category heuristic, the other heuristics shown in the table are infobox heuristics.

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

⁹Note that the significance depends only on the sample size. It is independent of the total size of the ontology.

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 *hasGDPPP* 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.

Comparison. It is difficult to compare YAGO to other information extraction approaches, because the approaches usually differ in the choice of relations and in the choice of the sources. Furthermore, precision can usually be varied at the cost of recall. Here, we just give an informational overview.

Approaches that use pattern matching (e.g. the Espresso System [105] or LEILA [133]) typically achieve precision rates of 50% to 92%, depending on the extracted relation. State-of-the-art taxonomy induction as described in [124] achieves a precision of 84%. KnowItAll [56] and KnowItNow [28] are reported to have precision rates of 85% and 80%, respectively. TextRunner [10] 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. This boils down to an overall accuracy of 57%.

Wu et al. [150] aim at filling in missing values in Wikipedia infoboxes and achieve a remarkable precision of 73% to 97%. The KOG ontology [151], using and improving upon the techniques of YAGO, is reported to achieve a precision of 96%. Ponzetto et al. [107] exploit the Wikipedia category network to construct a taxonomy and achieve a precision of around 87%. Banko et al. [11] use different domain search strategies for fact extraction and show a precision of around 80%.

Confidence Values. We have taken the precision estimates from the evaluation and attached them as confidence values to the facts. For example, all facts extracted by the WordNet linking technique have a confidence value of 95.12%. Many other information extraction approaches provide confidence values as well. However, in most cases, these are simply real valued scores between 0 and 1, which are hard to interpret. In YAGO, the confidence values have a precise meaning: The confidence value of a fact is the estimated probability that the fact is correct with respect to Wikipedia. This gives the confidence values a well-defined interpretation.

3.3.2 Size

Entities. Table 3 shows the number of entities in YAGO. Half of YAGO’s individuals are people and locations. Other prominent groups are institutions and movies. The overall number of entities is 2 million.

Table 3: Number of entities in YAGO

Relations	92
Classes	249,015
Individuals (without words and literals)	1,941,578
People	615,924
Locations	303,372
Institutions/companies	30,508
Movies	39,851

Facts. Table 4 shows the number of facts for the most frequent relations in YAGO. The overall number of ontological facts is 19 million. This number does not yet 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
means	5347523	hasSuccessor	55535
type	4505603	hasUTCOffset	52212
inLanguage	3563111	since	47714
isCalled	2185860	hasPopulationDensity	44628
familyNameOf	569410	produced	41747
givenNameOf	568852	hasProductionLanguage	40738
bornOnDate	441274	bornIn	36189
subClassOf	249463	hasImdb	33451
diedOnDate	205469	hasDuration	30791
hasWebsite	130098	actedIn	28836
establishedOnDate	110830	until	26049
isOfGenre	106797	directed	23723
created	95248	hasWonPrize	23076
hasPopulation	77928	writtenInYear	20663
hasArea	62720	hasPredecessor	20515
locatedIn	60261	musicalRole	15516

Appendix A.2 contains a complete list of relations.

Comparison. 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.

Table 5: Size of other ontologies

Ontology	# Entities	# Facts
SUMO [102]	20,000	70,000
Ponzetto et al. [107]	n/a	110,000
WordNet [59]	117,659	821,492
Cyc [95]	300,000	3,000,000
TextRunner [10]	n/a	7,800,000
YAGO	1,700,000	15,000,000
DBpedia [8]	1,950,000	116,000,000

3.3.3 Demonstration of Querying Capabilities

Simple Queries. As described in Section 3.2.4, 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? <i>Einstein hasDoctoralAdvisor ?x</i>	<i>?x=AlfredKleiner</i>
Who is named after a place in Africa? <i>?place locatedIn Africa</i> <i>?name means ?place</i> <i>?name familyNameOf ?who</i>	<i>?who=GabrielSudan</i> and 22 more

Advanced Queries. Table 7 shows three advanced queries. The first query uses a virtual relation (see Section 2.2.8) 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? <i>Canada hasHDI ?HDICanada</i> <i>?other hasHDI ?HDIother</i> <i>?HDIother > ?HDICanada</i>	<i>?other=Sweden</i> and 4 others
When did Angela Merkel become chancellor? <i>Angela Merkel type chancellor</i> <i>since ?when</i>	<i>?when=2005-11-22</i>
How is Germany called in Italian? <i>Germany isCalled ?how</i> <i>inLanguage Italian</i>	<i>?how=“Germania”</i>

Afterthoughts. It is tempting to assume some kind of “completeness” of YAGO and to ask, for example, 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. Put differently, it is tempting to assume that YAGO contains what is important. That is wrong. It contains what we could extract from Wikipedia.

3.4 Conclusion

Summary. This section has introduced the ontology YAGO. It forms the first component of the integrative knowledge gathering system that this thesis presents. The knowledge for YAGO has been extracted from Wikipedia. We showed how the category system and the infoboxes of Wikipedia can be exploited. We explained how Wikipedia and WordNet can be linked and how we can enforce high accuracy through type checks. Our evaluation proved 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.

Discussion. Although the knowledge extraction itself runs fully automated, 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 accuracy 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 possibly be applied to other sources as well. Many other sources, such as the catalogue of Amazon.com, the Internet Movie Database or the medical resource UMLS¹⁰, 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.

The following chapters will show how YAGO can be further extended in an automated way.

¹⁰<http://www.nlm.nih.gov/research/umls/>

Chapter 4

LEILA

The previous section has introduced the ontology YAGO, which forms the first component of our integrative knowledge gathering system. In order to extend YAGO, new information has to be added. This is done by the second component, the information extraction tool LEILA¹ [133, 134]. This chapter will first give an overview of the problem and related work. Then, it will present the model of LEILA. The last section shows our experiments with the system.

4.1 Overview

4.1.1 Problem Statement

This chapter discusses the task of information extraction (IE). This is the task of finding structured information in text documents. Here, we will concentrate on finding *facts* in the documents (in the sense of Section 1.3.2). For example, a text document might contain the following sentence

“Johannes Brahms was a German composer born in Hamburg.”

Then, the goal of a fact extraction system is to extract the following facts

<i>JohannesBrahms</i>	<i>type</i>	<i>composer</i>
<i>JohannesBrahms</i>	<i>hasNationality</i>	<i>German</i>
<i>JohannesBrahms</i>	<i>bornIn</i>	<i>Hamburg</i>

This task is non-trivial for a computer, because it requires the *understanding* of the document (see Section 1.3.1). More precisely, it requires the mapping of pieces of text to relations and facts. Information extraction can be pursued on different types of documents. Among them are structured documents (such as tables), semi-structured documents (such as Wikipedia, see Section 3.1.4.2) or unstructured natural language documents (such as news stories). Since a large part of content-bearing texts are unstructured in this sense, we aim at a system that can work on unstructured documents here. Furthermore, we aim at a system that works without human interaction. That is, our goal is to construct a system that, given a set of text documents, runs autonomously and extracts a set of facts. We will assume that the system is given a single target relation

¹Learning to Extract Information by Linguistic Analysis

(e.g., *bornInYear*). Then, the task is to find instances of the target relation in the documents (e.g., *FredericChopin/1810*).

4.1.2 Related Work

Numerous projects have pursued ways of extracting information from text documents. The approaches differ in various features and can hardly be classified strictly. The reader is invited to see [118] for a comprehensive survey of approaches. This section gives only a coarse categorization of existing approaches along several dimensions.

Type of Task. Some systems are designed to discover new relations ([90, 144]). In our setting, however, there is a given target relation (such as *bornInYear*). Some systems ([37, 160]) perform relation classification. For a given pair of entities, these systems try to choose the relation of which the pair is an instance. These systems may assume that the pair is indeed an instance of one relation out of a pool of given relations. In our setting, however, there may be pairs of entities that do not belong to any relation.

Type of the Relation. The extracted relation can be either unary or binary. In the unary case, the relations are just sets of entities (e.g. all *cities* in a given text, [60, 30]). The GATE project [45] offers (among other things) tools for recognizing certain types of entities, such as people or locations. The LIBRA project [161, 100] aims to identify canonic entities of a certain type in Web documents. In our setting, however, we focus on binary relations. Some systems aim at learning a single relation, mostly the *type*-relation ([43, 24, 64, 138, 41, 123, 42, 124, 115]). In this thesis, we are interested in extracting arbitrary relations. This includes not only the *type*-relation, but also other relations such as the *birthdate*-relation or the *headquarters*-relation between a company and the city of its headquarters.

Human interaction. There are systems that require human input for the IE process ([113, 92, 4, 91, 32]). The GATE project [45] also provides (among other things) a visual tool for human-oriented information extraction. Our work aims at a completely automated system.

Type of corpora. There exist systems that can extract information efficiently from formatted data, such as HTML-tables or structured text ([65, 61, 162]). As discussed in Section 3.1.2, there also exist very successful approaches that are tailored to specific corpora such as Wikipedia ([8, 107, 145]). However, since a large part of the Web consists of natural language text, we consider in this chapter only systems that accept also unstructured corpora. Furthermore, there exist approaches that concentrate on a specific domain, such as the biomedical domain ([109]) or the business domain ([117]). Here we aim at a domain-independent approach.

Initialization. As initial input, some systems require a hand-tagged corpus ([75, 126]), i.e., a corpus in which the relevant entities and relations have been marked manually. Other systems require text patterns ([155]) or templates ([153]), i.e., phrases that indicate an instance of the target relation. Again other

systems require seed tuples ([2, 20]), i.e., a set of instances of the target relation. Shen et al. [120] have proposed a combination of manually designed procedural predicates and logical reasoning to extract information (see Section 5.1.2). Since hand-labeled data, manually assembled text patterns and procedural predicates require huge human effort, we consider only systems that use seed tuples. Again other systems require the set of all individuals to be known a priori, which we may not assume here ([20], see Section 5.1.2).

Scope. Furthermore, we differentiate between *limited scope* systems that are bound to a corpus and *free scope* systems that use the Web as a corpus. Free scope systems are for example KnowItAll [56], TextRunner [10] (see Section 3.1.2) and Alice [11] (see Section 5.1.2). The DBLife project [52] pursues a slightly different goal, but still falls within the domain of free scope systems. We observe that in both types of systems, the techniques used to extract the entities from the documents are essential. These techniques will be the main focus of this chapter. To study the techniques in a controlled environment, we restrict ourselves to limited scope systems for this section.

Techniques. There are many different techniques for extracting entities from documents. One school concentrates on detecting the boundary of interesting entities in the text ([30, 60, 155]). Other approaches make use of the context in which an entity appears ([42, 25, 41]). This school is mostly focused on the *type-relation*. Another group of systems is the group of *pattern-oriented systems* ([56, 2, 111, 22, 125, 154, 11, 105, 143]). This classification is by no means complete or crisp; there exist numerous hybrid methods, which, for example, perform text segmentation but can be used for relation extraction. These include statistical methods that use Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs) (see again [118] for a more comprehensive survey).

Here, we provide a more detailed survey of pattern-oriented systems, because they are explicitly targeted at extracting facts with arbitrary relations.

Pattern-Oriented Systems. Pattern-oriented systems are given a target relation (e.g. *bornInLocation*) and a set of instances of the target relation (the *seed pairs*). They find patterns in the text that appear with a seed pair (e.g. the pattern “*X was born in Y*” might appear with a seed pair of the target relation *bornInLocation*). Then they seek other occurrences of that pattern and thereby find new instances of the target relation.

There are numerous techniques to extract patterns from text documents. Some approaches use raw surface patterns, i.e., they represent patterns as sequences of characters. Other systems use regular expressions as patterns, which allows them to generalize on patterns. Surprisingly, most existing systems do not use deep linguistic analysis of the corpus. Consequently, they are extremely volatile to small variations in the patterns – even if the variation does not have any semantic effect. For example, the simple subordinate clause in the following example (taken from [111]) can already prevent a surface pattern matcher from discovering the relation between “*London*” and the “*river Thames*”:

“*London, which has one of the busiest airports in the world, lies on the banks of the river Thames.*”

Furthermore, surface approaches cannot benefit from advanced linguistic techniques such as *anaphora resolution*. Anaphora resolution is the process of determining which entity is meant by a pronoun (such as “*he*” or “*she*”). The only approach that does use deep linguistic analysis [26] considers only the shortest path in the dependency graph as a feature. Thus, this system cannot deal with the difference between “*A dog is a mammal*” (which expresses the *sub-ClassOf*-relation) and “*This dog is a nag*” (which does not). Most importantly, all pattern matching approaches can be misled by *false positive patterns*, i.e., patterns that appear with a seed pair, but that are not indicative of the target relation. However, the exact influence of false positive patterns has rarely been analyzed.

4.1.3 Contribution

This section presents LEILA [133, 134], a pattern-oriented system with novel techniques for fact extraction from text documents. Given a binary target relation (such as *bornInLocation*), LEILA will extract new instances of the relation from a text corpus. LEILA brings three key contributions:

1. **Linguistic Analysis:** LEILA uses a link-grammar representation [122] for natural-language sentences. This allows LEILA to detect robust natural language patterns. Furthermore, it allows for advanced techniques such as anaphora resolution.
2. **Counterexamples:** LEILA takes into account counterexamples, i.e., pairs of entities that are known to be not in the target relation. This allows LEILA to identify and discard false positive patterns. Discarding these patterns improves LEILA’s precision.
3. **Machine Learning:** LEILA uses statistical learning (namely SVMs and kNN classifiers) to generalize the useful patterns. This process gives LEILA high yield and robust patterns, increasing its recall and precision.

As discussed in the previous section, there are existing works that also use linguistic analysis [26, 123]. There are also numerous previous works that use machine learning techniques ([123] among others). However, none of the previous works uses machine learning and linguistic analysis to extract *arbitrary* relations. Furthermore, none of the previous approaches uses counterexamples in addition to the seed pairs.

All of LEILA’s techniques are carefully integrated into a full-fledged system architecture. Our evaluation shows that LEILA outperforms state-of-the-art techniques for information extraction. Our theoretical analysis shows that false positive patterns cannot disrupt the performance of LEILA.

4.1.4 Linguistic Analysis

Linguistic Structures. LEILA sees sentences not as sequences of words, but as deep linguistic structures. The process of constructing a deep linguistic structure for a sentence is called *parsing*. There exist different approaches for parsing. They range from simple part-of-speech tagging to context-free grammars and

more advanced techniques such as Lexical Functional Grammars, Head-Driven Phrase Structure Grammars or stochastic approaches. In principle, LEILA can work with any type of linguistic structures.

Link Parser. For purely pragmatic reasons, we chose the structures produced by the Link Grammar Parser [122]. This parser is based on a context-free grammar and it is simpler to handle than the advanced parsing techniques. At the same time, it provides a much deeper semantic structure than the standard context-free parsers. Following our implementation choice, this subsection defines the notions of *linguistic structure* and *pattern* with respect to the Link Grammar Parser, even though any other parser could be used.

Linkages. Figure 5 shows a linguistic structure produced by the Link Parser. We call these structures *linkages*.

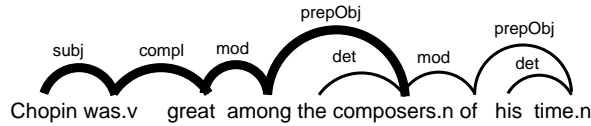


Figure 5: A simple linkage

Technically speaking, a linkage is a connected planar undirected graph, the nodes of which are the words of the sentence. The edges (the *links*) are labeled with *connectors*. For example, the connector *subj* marks the link between the subject and the verb of the sentence. The linkage must fulfill certain linguistic constraints. These are given by a *link grammar*, which specifies which word may be linked by which connector to preceding and following words. The parser also determines the *part-of-speech* (POS) for the words. The POS of a word can be noun, verb, adjective, adverb or preposition. In Figure 5, the suffix “.n” identifies “*composers*” as a noun.

Patterns. We define patterns based on linkages:

DEFINITION 17: [*Pattern*]

A *pattern* is a linkage in which two words have been replaced by placeholders.

Figure 6 shows a sample pattern with the placeholders “X” and “Y”.

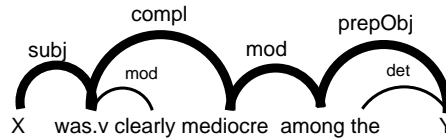


Figure 6: A simple pattern

We call the (unique) shortest path from one placeholder to the other the *bridge*, marked in bold in Figure 6. A pattern *matches* a linkage if the bridge of the pattern appears in the linkage, although nouns and adjectives are allowed to differ. For example, the pattern in Figure 6 matches the linkage in Figure 5, because the bridge of the pattern occurs in the linkage, apart from a substitution

of “great” by “mediocre”. If a pattern matches a linkage, we say that the pattern *produces* the pair of words that the linkage contains in the position of the placeholders. In our example, the pair “Chopin” / “composers” is produced.

Semantics. We say that a pattern *expresses* a relation r , if the underlying sentence implies that a pair of entities is an instance of r . Note that the deep linguistic analysis of the sentence would allow us to define the meaning of the sentence in a theoretically well-founded way [98]. For this thesis, however, we limit ourselves to an intuitive understanding of the notion of meaning. We make another simplifying assumption: Strictly speaking, patterns produce pairs of words, not pairs of entities. The words have to be mapped to entities in order to check whether the pair is an instance of a relation. In tune with all other pattern matching approaches, we will ignore this problem for this section. We will assume that there is a many-to-one mapping between words and entities. Thus, we will say that the pair produced by a pattern is an element of the target relation even though the pair of entities is meant. Section 5 will deal with the problem of mapping words to entities.

4.2 System Model

4.2.1 Preprocessing

LEILA will find patterns and relation instances in a corpus. To facilitate this task, we preprocess the corpus.

Character Normalization. Different sources come with different encodings for special characters (such as umlauts). Some sources use ampersand codes (such as “ä”), others use percentage codes (such as “%A3”), again others use backslash codes (such as “\u00A3”) and again others use UTF-8. We have developed a parser that decodes these types of codes to Unicode characters.² Some characters are non-ASCII Unicode characters and hence hard to digest for the Link Parser. Hence, our parser can also normalize the characters. Normalization maps a Unicode character to an ASCII character (or to a sequence thereof). Of course, our normalization cannot cover all Unicode characters, but our experience shows that Unicode characters not covered by our normalization hardly appear.

Sentence Detection. Our preprocessor eliminates formatting elements from the corpus. This includes HTML tags in HTML files and blank lines and repeated spaces in text files. The result of this process is corpus of pure text. Then, the preprocessor splits the corpus into sentences. It uses simple heuristics (such as occurrences of punctuation) for this purpose. Some parts of the corpus are ungrammatical. These parts are lists of items, expressions that use parentheses and other constructions that cannot be handled by the Link Parser. Our preprocessor cuts them out. Consider, for example, the sentence “Chopin and Mozart (and others) were composers”. The preprocessor cuts out “(and others)”. This leaves the proper sentence “Chopin and Mozart were composers”

²All tools mentioned here are available as *Java Tools* at <http://mpii.de/~suchanek/downloads/javatools>.

and produces the non-grammatical “sentence” “*Chopin and Mozart (and others)*”. The non-grammatical sentence is given a pseudo-linguistic parsing by the preprocessor, which just links adjacent words by an artificial connector.

Date and Number Normalization. Many relations concern values such as dates and numbers. These values appear in various forms in natural language texts. For example, the following strings are valid expressions of “100 meters”:

100 meters, 100 m, one hundred meters, a hundred meters, 0.1 km, 1000 decimeters, 0.062 miles, 328 feet, 328'

A similar variety exists for dates. For example, the following strings are valid expressions for “December 19th, 2008”:

2008-12-19, 12/19/2008, 19.12.2008, December 19th 2008, 19th of December 2008, 19-Dec-2008

We have developed a parser that normalizes all of these expressions. After the normalization, all dates are in ISO 6008 format (i.e., of the form “2008-12-19”). Partial date expressions (such as “May 2008”) are encoded with placeholders (“2008-05-##”). Analogously, all number expressions take the form of digits plus an optional unit identifier (“100m”). Non-SI units (such as gallons, stones, Fahrenheit and acre-feet) are converted to SI units.

Stemming. To facilitate the detection of seed pairs, words have to be in their singular form. Stemming plural words to singular words is a non-trivial enterprise. Some words form their singular by cutting off an ‘s’, whereas others cut off ‘es’ (consider, e.g., “boxes”/“box”, but “nurses”/“nurse”). Some words have a foreign origin and hence form their singular completely differently (such as “automata”/“automaton”). Some words have a completely irregular form (“mice”/“mouse”). Other words do not change at all (such as “atlas”, which can be both singular and plural). Again others look like a plural word, but are not (“aerobics”). Worse, some words can be both plural and singular (such as “mechanics” as the discipline or the plural of “mechanic”). By collecting these exceptions systematically from WordNet [59], we were able to develop a stemmer (the *PlingStemmer*) that can handle most cases in the English language.

4.2.2 Algorithm

Target Relation. As input, LEILA requires the target relation, i.e., the relation for which the system shall find instances. As a definition of the target relation, LEILA expects a function that decides into which of the following categories a pair of words falls:

- **Example:** The pair is an instance of the target relation.
- **Counterexample:** The pair is not an instance of the target relation.
- **Candidate:** The pair could be an instance of the target relation.
- None of the above.

Corpus. Furthermore, LEILA requires a preprocessed corpus as input. The sentences in the corpus are parsed, producing a deep *linguistic structure* for each of them. In principle, our algorithm does not depend on a specific parsing technique. In our implementation, we use the Link Grammar Parser. The next step is an optional *anaphora resolution*, i.e., the replacement of pronouns by their references. We use a conservative approach, which simply replaces a third person pronoun by the subject of the preceding sentence.

Algorithm. Summing up, we are given as input

1. A target relation in the form of a function, as described above.
2. A preprocessed and parsed corpus.

As output, we aim to produce a set of instances of the target relation (see Section 1.3.2). LEILA’s core algorithm proceeds in three phases:

1. In the *Discovery Phase*, it seeks sentences and corresponding linkages in which an example pair appears. It replaces the two words by placeholders, thus producing a pattern. These patterns are collected as *positive patterns*. Then, the algorithm runs through the sentences again and finds all linkages that match a positive pattern, but produce a counterexample. The corresponding patterns are collected as *negative patterns*³.
2. In the *Training Phase*, statistical learning is applied on the positive and negative patterns. The result of this process is a classifier for patterns.
3. In the *Testing Phase*, the algorithm considers again all sentences in the corpus. For each linkage, it generates all possible patterns by replacing two words by placeholders. If the two words form a candidate and the pattern is classified as positive, the produced pair is proposed as a new element of the target relation (an *output pair*).

This way, the system generates new instances of the target relation from the corpus.

Example. Suppose our target relation is *bornInYear*, i.e., we are interested in people and their birth dates. For LEILA, we have to characterize this relation by a function. This function has to decide whether a pair of words is an example, a counterexample or a candidate. For *bornInYear*, this function could make use of a set of known pairs of people with their birth dates (the *seed pairs*). For example, the seed pairs could contain the pair *Chopin/1810*. Then, the function categorizes a pair of words x/y as follows:

- The pair is an **example**, if it appears in the seed pairs. In our case, the pair with $x = \textit{Chopin}$ and $y = 1810$ is an example.

³While *counterexamples* are pairs of entities that do not stand in the target relation, *negative patterns* are pieces of text that do not express the target relation. Note that different patterns can match the same linkage.

- The pair is a **counterexample**, if x is a person appearing in the seed pairs and y is a wrong birth date. For example, the pairs *Chopin/1811*, *Chopin/1812*, and *Chopin/2000* would all be categorized as counterexamples, because we know that Chopin was born in 1810.
- A pair is a **candidate**, if x is a proper name, y is a year and the pair is neither an example nor a counterexample. For example, assuming that our seed pairs know nothing about Mozart, the following pairs are candidates: *Mozart/1950*, *Mozart/1870*, *Mozart/1756*
- In all other cases, the pair is classified as **none**.

Our algorithm requires a parsed corpus as input. For the sake of simplicity, we will represent the corpus and the patterns as plain sentences for our example (see Section 4.1.4 for the exact form of patterns). The algorithm proceeds as follows:

1. In the *Discovery Phase*, it seeks sentences and corresponding linkages in which an example pair appears. For example, assume that the corpus contains the following sentence

“Chopin was born in 1810.”

Since the pair *Chopin/1810* is classified as an example pair by our function, the algorithm generates the following pattern

“X was born in Y.”

Now suppose we find the following sentence

“Chopin has composed more than 1810 pieces of music.”

In this sentence, the example pair *Chopin/1810* appears as well. Hence, we also record the following pattern as a positive pattern:

“X has composed more than Y pieces of music.”

If this last pattern also occurs with counterexamples, the algorithm will register the pattern both as a positive pattern and as a negative pattern.

2. In the *Training Phase*, statistical learning is applied on the positive and negative patterns. The result of this process is a classifier for patterns. In our case, the pattern

“X was born in Y.”

will be classified as positive. The pattern

“X has composed more than Y pieces of music.”

will hopefully have appeared with many counterexamples, so that it is classified as negative. We discuss in Section 4.2.3 what happens if this pattern is erroneously classified as positive.

3. In the *Testing Phase*, the algorithm considers again all sentences in the corpus. For example, assume that we find the sentence

“Mozart was born in 1756.”

Since the pair *Mozart/1756* is categorized by our target relation function as a candidate, we examine the following pattern:

“X was born in Y.”

Since this pattern is classified as positive by our pattern classifier, the algorithm produces the pair *Mozart/1756* as an output pair.

Finally, the output of the algorithm is a set instances of the target relation *bornInYear*. Although usually the Discovery Phase and the Testing Phase are run on the same corpus, it is also possible to run them on two distinct corpora.

4.2.3 Robustness

4.2.3.1 The Problem of False Samples

False Samples. The central task of the Discovery Phase is determining patterns that express the target relation. Since the linguistic meaning of the patterns is not apparent to the system, it relies on the following hypothesis: Whenever an example pair appears in a sentence, the linkage expresses the target relation. This hypothesis may fail if a sentence contains an example pair merely by chance, i.e., without expressing the target relation. For example, assume that we are again considering the *bornInYear* relation. Assume that the pair *Chopin/1810* is a seed pair. Then, the following sentence would trigger the extraction of a positive pattern in the Discovery Phase:

“Chopin has composed more than 1810 pieces of music.”

Clearly, this sentence does not express our target relation *bornInYear*. Still, the pattern would be used as a positive sample for the generalization process. Analogously, a pattern that does express the target relation may occasionally produce counterexamples. For example, the pattern *“X was born in Y”* does express our target relation *bornInYear*, but still it produces a counterexample if applied to the following sentence:

“Chopin was born in 1811 or 1810.”

In this case, the pattern is used as a negative sample in the generalization process, even though it does express the target relation. We call these patterns *false samples*. The problem of false samples is intrinsic for pattern matching approaches in general. We will now see why false samples do not question the validity of our approach.

False Samples in Machine Learning. Virtually any learning algorithm can deal with a limited number of false samples. For Support Vector Machines (SVM), the effect of false samples has been analyzed thoroughly in [39]. In

general, an SVM is highly tolerant to noise. There are also detailed theoretical studies [7] on how the proportion of false samples influences a PAC-learner. In essence, the number of required samples increases, but the classification is still learnable.

It is also possible to understand the concept of positive patterns as a probabilistic concept [83]. In this setting, the pattern is not classified as either positive or negative, but it may produce pairs of the target relation with a certain fixed probability. The task of the learner is to learn the function from the pattern to its probability. [121] shows that probabilistic concepts can be learned and gives bounds on the number of required samples.

The following subsection analyzes the problem of false samples in more detail for a special class of classifiers, the k-Nearest-Neighbor-classifiers.

4.2.3.2 False Samples with k-Nearest-Neighbor Classifiers

kNN Classifiers. A k-Nearest-Neighbors (kNN) classifier takes as input a set of *training patterns* that are labeled positive or negative. In our case, this set is provided incrementally during the Discovery Phase. In addition to the training patterns, the kNN classifier also requires a distance function on patterns. In the Testing Phase, the classifier can be asked to classify a (previously unknown) *test pattern* as positive or negative. It does this by considering the distance of the test pattern to the training patterns.

Adaptive kNN Classifiers. We consider a simple variant of an adaptive kNN classifier: During the Discovery Phase, a newly added positive pattern becomes a *prototype* for a whole class of new patterns. Whenever another positive pattern is discovered, we check whether its distance to an existing prototype is below some threshold θ . We say that the pattern *falls on* the prototype⁴. If the new pattern does not fall on an existing prototype, it becomes a prototype on its own. The actual Training Phase is particularly simple for the kNN Classifier: We label a prototype as *positive* if the majority of the patterns that fell on it were positive, as *negative* else. In the Testing Phase, we find for a test pattern its closest prototype. If there is no prototype within the distance θ , the pattern is classified as negative. If it falls on a prototype p , the pattern is classified as positive if p has a positive label and as negative else.

Probabilistic Model. We are interested in the probability that a test pattern is classified as positive, although the produced pair is not in the target relation. In the Testing Phase, each possible pattern is generated for each sentence in the corpus (this will be a number of patterns quadratic in the number of nouns in the sentence). We model the sequence of all these patterns as a sequence of N random events. Each pattern produces a pair of words with its underlying sentence. This pair can either be an example, a counterexample or a candidate⁵. We model these events by Bernoulli random variables $EX, CE, CAND$,

⁴If θ is chosen sufficiently small, all patterns falling on p share their essential linguistic properties. Hence we assume that they all have the same probability of producing examples or counterexamples.

⁵For simplification, we assume that the 4th class of word pairs mentioned in Section 4.2.2 does not appear. If it does, it will only improve the bound given here.

captured by a multinomial distribution: $EX = 1$ iff the pair is an example, $CE = 1$ iff the pair is a counterexample, $CAND = 1 - EX - CE = 1$ iff the pair is a candidate. For each prototype p , we introduce a Bernoulli random variable F_p , such that $F_p = 1$ with probability f_p iff a generated pattern falls on p . Note that this model also applies to the Discovery Phase.

Training Phase. We first concentrate on the Training Phase. We are interested in the probability that a given prototype p gets a positive label, although it does not express the target relation. We define the *quality* of p as the relative probability of patterns falling on p to produce examples:

$$q_p = \frac{P(EX|F_p)}{P(EX|F_p) + P(CE|F_p)}$$

Since p does not express⁶ the target relation, $q_p < \frac{1}{2}$. The *allotment* of p is the share of examples and counterexamples produced by patterns falling on p :

$$a_p = P(EX|F_p) + P(CE|F_p)$$

The better the examples and counterexamples are chosen, the more likely it is that patterns falling on p produce examples or counterexamples (instead of candidates) and the larger a_p will be. Let $\#EX_p$ stand for the number of examples and $\#CE_p$ for the number of counterexamples produced by patterns falling on p in the Discovery Phase. We are interested in the probability of p getting a positive label, namely $P(\#EX_p > \#CE_p)$, given that $q_p < \frac{1}{2}$. In Appendix B.3 we prove the following theorem:

THEOREM 3: [*Probability of False Labeling*]

With the above definitions, the probability that a prototype p receives a positive label in the training phase of the adaptive kNN classifier is bounded as follows:

$$P(\#EX_p > \#CE_p) \leq 2e^{-\frac{1}{2}Na_p^2f_p^2} + 2e^{(2-a_p f_p N) \cdot (\frac{1}{2}-q_p)^2}$$

Testing Phase. Now we turn to the Testing Phase. We are interested in the probability that an incorrect output pair is produced by a pattern falling on p . For this to happen, a test pattern must fall on p , it must produce a candidate and p must be wrongly labeled as positive. Combined, this yields

$$\begin{aligned} & P(CAND \cap F_p) \cdot P(\#EX_p > \#CE_p) \\ = & P(CAND|F_p) \cdot P(F_p) \cdot P(\#EX_p > \#CE_p) \\ = & (1 - a_p) \cdot f_p \cdot P(\#EX_p > \#CE_p) \\ \leq & 2(1 - a_p) \cdot f_p \cdot (e^{-\frac{1}{2}Na_p^2f_p^2} + e^{(2-a_p f_p N) \cdot (\frac{1}{2}-q_p)^2}) \end{aligned}$$

This estimation shows that a larger allotment a_p (i.e., a good choice of examples and counterexamples) decreases the probability of wrongly classifying a candidate pair. Furthermore, the estimation mirrors the intuition that either many

⁶Note that if $q_p > \frac{1}{2}$, the pattern does express the target relation and there is no danger that it could produce false output pairs. Here, we concentrate on patterns that do not express the target relation, but are erroneously assumed to express the target relation by the algorithm. Hence, the interesting case is $q_p < \frac{1}{2}$.

patterns fall on p in the Discovery Phase (f_p large) and then p is unlikely to have a false label, or few patterns fall on p (f_p small) and then the probability of p classifying a test pattern is small. As the number of sentences (and hence the number of generated patterns N) increases, the bound converges to zero. This observation entails the following corollary of Theorem 3:

COROLLARY 2: [*Probability of False Output Pairs*]

If an adaptive kNN classifier is used, the probability that LEILA produces an output pair that is not in the target relation converges to zero as the number of patterns increases.

This insight provides the theoretical justification for our approach.

4.2.4 Classifying Patterns

This subsection discusses how patterns can be represented and generalized using machine learning. We will first identify the important features of patterns. Then, we will show how the kNN classifier and the SVM classifier can work on these features.

4.2.4.1 Feature Model

Feature Vectors. Most machine learners require that the patterns are represented as a set of *features*. A feature is a certain characteristics of the pattern. For example, one feature could be the number of words in the pattern. It would not be too difficult to somehow encode the full patterns into features. However, such an approach would not generalize well, for it would capture all details of the specific sentences that led to the patterns and thus tend to cause overfitting. So the problem that we tackle is to identify the characteristic but generalized features within linkages.

Bridges and Contexts. The most important component of a pattern is its bridge. In the Discovery Phase, we collect the bridges of the patterns in a list. Each bridge is given an identification number, the *bridge id*. Two bridges are given the same bridge id if they differ only in their nouns or adjectives (as discussed in section 4.1.4). The *context* of a word in a linkage is the set of all its links together with their direction in the sentence (left or right) and their target words. For example, consider again the linkage given in Figure 7.

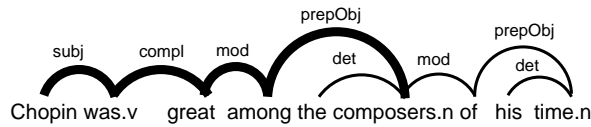


Figure 7: A simple linkage

The context of “*composers*” in this linkage is the following set of triples

(*det*, left, “*the*”)
 (*prepObj*, left, “*among*”)
 (*mod*, right, “*of*”)

Word Types. Each word is assigned a set of *types*. We distinguish nouns, adjectives, prepositions, verbs, numbers, dates, names, person names, company names and abbreviations. The parser already assigns the grammatical types by its part-of-speech tagging. We assign the other types by regular expression matching. For example, any word matching “[A-Z]/[a-z]+ Inc” is given the type *company*. To accommodate the role of stopwords in understanding a sentence, we make each stopword a type of its own.

Features of a Pattern. We represent a pattern by the following set of features: The bridge id, the context of the first placeholder and the context of the second placeholder. For example, supposing that the bridge id of the pattern in Figure 7 is 42, we represent the pattern as

bridge id:	42
first context:	$\{(subj, right, "was")\}$
second context:	$\{(det, left, "the"),$ $(prepObj, left, "among"),$ $(mod, right, "of")\}$

Each pattern is reduced to its features before it is given to the machine learning algorithm. Furthermore, each pattern found in the Discovery Phase is equipped with a *label* (+1 for positive patterns and −1 for negative patterns). To show that our approach does not depend on a specific learning algorithm, we implemented two machine learning algorithms: The adaptive kNN classifier discussed in 4.2.3.2 and an SVM classifier. Since the kNN classifier requires a distance function, whereas the SVM classifier requires a vector representation of patterns, both classifiers use slightly different inputs.

4.2.4.2 Feature Model for the kNN Classifier

Similarity Functions. For the adaptive kNN, we need a similarity function on patterns. Let $\tau(w)$ be the set of types of a word w . The similarity of two words is the overlap of their type sets:

$$sim(w_1, w_2) = \frac{|\tau(w_1) \cap \tau(w_2)|}{|\tau(w_1) \cup \tau(w_2)|}$$

The similarity of two contexts C_1, C_2 is computed by comparing each triple in C_1 to all triples in C_2 , where each triple contains a connector, a direction and a word:

$$sim(C_1, C_2) = \sum_{(con_i, dir_i, w_i) \in C_i, i=1,2} \frac{\alpha_1([con_1 = con_2]) + \alpha_2([dir_1 = dir_2]) + \alpha_3 sim(w_1, w_2)}{|C_1| \cdot |C_2|}$$

Here, $[\cdot]$ denotes the Iverson bracket, which evaluates to 1 if the enclosed condition is true and to 0 else. $\alpha_1, \alpha_2, \alpha_3$ are weighting factors that sum up to 1. We chose $\alpha_1 = 0.4, \alpha_2 = 0.2, \alpha_3 = 0.4$. Two patterns have a similarity of zero if they have different bridge ids. Else, their similarity is the averaged similarity

of the contexts of the first and second placeholder, respectively:

$$\text{sim}((b_1, C_{11}, C_{12}, l_1), (b_2, C_{21}, C_{22}, l_2)) = \frac{1}{2}[b_1 = b_2](\text{sim}(C_{11}, C_{21}) + \text{sim}(C_{12}, C_{22}))$$

Training and Testing. Let c_P be the set of patterns that fell on a prototype P during the Discovery Phase. We compute the label of P as the sum of the labels of the patterns in c_P , weighted with their respective similarities to P :

$$\text{label}(P) = \sum_{p \in c_P} \frac{\text{label}(p) \cdot \text{sim}(P, p)}{|c_P|}$$

To classify a pattern p in the Testing Phase, we first determine its prototype P . If there is no prototype within the distance θ , the pattern receives the label $-\infty$. Else, we calculate its label as the product of the similarity to the prototype and the label of the prototype.

$$\text{label}(p) = \text{label}(P) \cdot \text{sim}(P, p)$$

If the label is greater than zero, p is classified as positive. Else, p is considered negative.

4.2.4.3 Feature Model for the SVM Classifier

Feature Vectors. To generalize patterns by an SVM, the patterns have to be translated to real-valued feature vectors. For this purpose, we first group the patterns by their bridge ids. Each group will be treated separately so that it is not necessary to store the bridge id in the feature vector. If n is the number of connector symbols, then a feature vector for a pattern can be depicted as follows:

$$\begin{array}{ccccccc} & \text{context 1} & & & \text{context 2} & & \\ \underbrace{X \dots X}_{\text{connector}_1} & \dots & \underbrace{X \dots X}_{\text{connector}_n} & \underbrace{X \dots X}_{\text{connector}_1} & \dots & \underbrace{X \dots X}_{\text{connector}_n} \end{array}$$

The vector consists of two parts, which store the context of the first and second placeholder, respectively. For example, consider again the representation of the sample pattern introduced in the previous section:

bridge id:	42
first context:	$\{(subj, right, "was")\}$
second context:	$\{(det, left, "the"),$ $(prepObj, left, "among"),$ $(mod, right, "of")\}$

The first context of this representation will be stored in the first part of the feature vector. The second context will be stored in the second part of the feature vector. Let us examine how a context can be stored in the vector: In the vector, each context area contains a sub-part for each possible connector symbol. For example, there will be one sub-part for *subj*, one sub-part for *obj* and so on. Each of these subparts contains one bit (denoted by X in the above scheme) for each possible word type. So, in each sub-part, there will be one

bit for the type noun, one bit for the type verb etc. Hence, if there are t word types, the overall length of the vector is $1 + n \cdot t + n \cdot t$. We encode a context as follows in the vector: If there is a link with connector *con* that points to a word w , we first select the sub-part that corresponds to the connector symbol *con*. Within this sub-part, we set all bits to 1 that correspond to a type that w has. For example, assuming that there were only 4 word types and assuming that *verb* is the first type, the first context of our sample pattern is encoded as

$$\overbrace{\underbrace{0, 0, 0, 0}_{\text{connector}_1} \dots \underbrace{1, 0, 0, 0}_{\text{subj}} \dots \underbrace{0, 0, 0, 0}_{\text{connector}_n}}^{\text{context 1}}$$

Training Phase. During the Discovery Phase, the system produces feature vectors for positive and negative patterns. These vectors are always grouped according to the bridges. After the Discovery Phase, we pass each group separately to an SVM. We used SVMlight [79] with its default parameters. The SVM produces a *model* for each group, i.e., basically a function from patterns to real values (negative values for negative patterns and positive values for positive ones).

Testing Phase. To classify a new pattern in the Testing Phase, we first identify its bridge group. If the pattern does not belong to a known group, we give it the label $-\infty$. Else, we translate the pattern to a feature vector and then apply the model of its group. Note that both the kNN classifier and the SVM classifier output a real value that can be interpreted as the confidence of the classification. Thus, it is possible to rank the output pairs by their confidence.

4.3 Experiments

4.3.1 Setup

Corpora. We ran LEILA on different corpora with increasing heterogeneity:

- **Wikicomposers:** The set of all Wikipedia articles about composers (872 HTML documents). We use it to see how LEILA performs on a document collection with a strong structural and thematic homogeneity.
- **Wikigeography:** The set of all Wikipedia pages about the geography of countries (313 HTML documents).
- **Wikigeneral:** A set of random Wikipedia articles (78141 HTML documents). We chose it to assess LEILA’s performance on structurally homogenous, but thematically random documents.
- **Googlecomposers:** This set contains one document for each baroque, classical, and romantic composer in Wikipedia’s list of composers, as delivered by a Google ”I’m feeling lucky” search for the composer’s name (492 HTML documents). We use it to see how LEILA performs on a corpus with a high structural heterogeneity. Since the querying was done

automatically, the downloaded pages include spurious advertisements as well as pages with no proper sentences at all.

Target Relations. We tested LEILA on different target relations with increasing complexity:

- **birthdate:** This relation holds between a person and his birth date (e.g. “*Chopin*” / “*1810*”). It is easy to learn, because it is bound to strong surface clues (the first element is always a name, the second is always a date).
- **synonymy:** This relation holds between two names that refer to the same entity (e.g. “*UN*” / “*United Nations*”). The relation is more sophisticated, since there are no surface clues.
- **type** This relation is even more sophisticated, because the sentences often express it only implicitly.

Competitors. We compared LEILA to different competitors. We only considered competitors that, like LEILA, extract the information from a corpus without using other Internet sources. We wanted to avoid running the competitors on our own corpora or on our own target relations, because we could not be sure to achieve a fair tuning of the competitors. Hence we ran LEILA on the corpora and the target relations that our competitors have been tested on by their authors. We compare the results of LEILA with the results reported by the authors. Our competitors, together with their respective corpora and relations, are:

- **TextToOnto:**⁷ A state-of-the-art representative for non-deep pattern matching. The system provides a component for the *type* relation and takes arbitrary HTML documents as input. For completeness, we also consider its successor Text2Onto [42], although it contains only default methods in its current state of development.
- **Snowball [2]:** A recent representative of the slot-extraction paradigm. In the original paper, Snowball has been tested on the *headquarters* relation. This relation holds between a company and the city of its headquarters. Snowball was run on a collection of some thousand documents. For the evaluation, the authors extracted the ground truth manually from a subset of this corpus. For copyright reasons, we only had access to this sub-corpus. It consists of 150 text documents.
- [43] present a new system that uses context to assign a concept to an entity. We will refer to this system as the **CV-system**. The approach is restricted to the *type*-relation, but it can classify instances even if the corpus does not contain explicit definitions. In the original paper, the system was tested on a collection of 1880 files from the Lonely Planet Internet site⁸.

⁷<http://www.sourceforge.net/projects/texttoonto>

⁸<http://www.lonelyplanet.com/>

Metrics. For the evaluation, the output pairs of the system have to be compared to a table of ideal pairs. One option would be to take the ideal pairs from a pre-compiled data base. The problem is that these ideal pairs may differ from the facts expressed in the documents. Furthermore, these ideal pairs do not allow to measure how much of the document content the system actually extracted.

This is why we chose to extract the ideal pairs manually from the documents. In our methodology, the ideal pairs comprise all pairs that a human would understand to be elements of the target relation. This involves full anaphora resolution, the solving of reference ambiguities, and the choice of truly defining concepts. For example, we accept Chopin as instance of *composer* but not as instance of *member*, even if the text says that he was a member of some club. Of course, we expect neither the competitors nor LEILA to achieve the results in the ideal table. However, this methodology is the only fair way of manual extraction, as it is guaranteed to be system-independent. If O denotes the multi-set of the output pairs and I denotes the multi-set of the ideal pairs, then precision, recall, and their harmonic mean $F1$ can be computed as

$$\begin{aligned} recall &= \frac{|O \cap I|}{|I|} & precision &= \frac{|O \cap I|}{|O|} \\ F1 &= \frac{2 \times recall \times precision}{recall + precision} \end{aligned}$$

Additional Metrics. To ensure a fair comparison of LEILA to Snowball, we use the same evaluation as employed in the original Snowball paper [2], the *Ideal Metric*. The Ideal Metric assumes the target relation to be functional. Hence the set of ideal pairs is right-unique. The set of output pairs can be made right-unique by selecting the pair with the highest confidence for each first component. Duplicates are removed from the ideal pairs and also from the output pairs. All output pairs that have a first component that is not in the ideal set are removed. There is one special case for the CV-system, which uses the Ideal Metric for the non-right-unique *type* relation. To allow for a fair comparison, we used the *Relaxed Ideal Metric*, which does not make the ideal pairs right-unique. The calculation of recall is relaxed as follows:

$$recall = \frac{|O \cap I|}{|\{x | \exists y : (x, y) \in I\}|}$$

By taking over the metrics from our competitors, we ensure that the competitors are evaluated under optimal conditions.

Significance. Due to the effort, we could extract the ideal pairs only for a sub-corpus. To ensure significance in spite of this, we compute confidence intervals for our estimates: We interpret the sequence of output pairs as a repetition of a Bernoulli-experiment, where the output pair can be either correct (i.e., contained in the ideal pairs) or not. The parameter of this Bernoulli-distribution is the precision. We estimate the precision by drawing a sample (i.e., by extracting all ideal pairs in the sub-corpus). By assuming that the output pairs are identically independently distributed, we can calculate a confidence interval for

our estimation. We compute confidence intervals for precision and recall for a confidence level of $\alpha = 95\%$. We measure precision at different levels of recall and report the values for the best F1 value.

We used approximate string matching techniques to account for different writings of the same entity. For example, we count the output pair “Chopin” / “composer” as correct, even if the ideal pairs contain “Frederic_Chopin” / “composer”. To ensure that LEILA does not just reproduce the example pairs, we list the percentage of examples among the output pairs. During our evaluation, we found that the Link Grammar parser does not finish parsing on roughly 1% of the files for unknown reasons.

4.3.2 Results

4.3.2.1 Results on different relations

Table 8 summarizes our experimental results with LEILA on different relations. All confidence intervals for precision and recall are below $\pm 10\%$ (see [133] for the values).

Table 8: Results with different relations

Corpus	Rel.	#D	#O	#C	#I	Precision	Recall	F1	%E
Wikicomposers	birthd.	87	95	70	101	73.68%	69.31%	71.43%	4.29%
Wikicomposers	birthd.	87	90	70	101	78.89%	70.30%	74.35%	4.23%
Wikigeography	syn.	81	92	74	164	80.43%	45.12%	57.81%	5.41%
Wikigeography	syn.	81	143	105	164	73.43%	64.02%	68.40%	4.76%
Wikicomposers	type	87	685	408	1127	59.56%	36.20%	45.03%	6.62%
Wikicomposers	type	87	790	463	1127	58.61%	41.08%	48.30%	7.34%
Wikigeneral	type	287	921	304	912	33.01%	33.33%	33.17%	3.62%
Googlecomposers	type	100	787	210	1334	26.68%	15.74%	19.80%	4.76%
Googlecomposers	type	100	840	237	1334	28.21%	17.77%	21.80%	8.44%
Googlec.+Wikic.	type	100	563	203	1334	36.06%	15.22%	21.40%	5.42%
Googlec.+Wikic.	type	100	826	246	1334	29.78%	18.44%	22.78%	7.72%

#D – number of documents in the hand-processed sub-corpus

#O – number of output pairs

#C – number of correct output pairs

#I – number of ideal pairs

%E – proportion of example pairs among the correct output pairs

Birthdate. For the birthdate relation, we used Edward Morykwas’ list of famous birthdays⁹ as examples. As counterexamples, we chose all pairs of a person that was in the examples and an incorrect birthdate. All pairs of a proper name and a date are candidates. We ran LEILA on the Wikicomposer corpus. LEILA performed quite well on this task. The patterns found were of the form “X was born in Y” and “X (Y)”. The quality of the results decreases as the system starts to consider any number in brackets a birthdate. For example, at the lower end of the confidence scale, the system also reports operas with the date of their first performance.

Synonymy. For the synonymy relation we used all pairs of proper names that share the same synset in WordNet as examples (e.g. “UN” / “United Nations”). As counterexamples, we chose all pairs of nouns that are not synonymous in

⁹<http://www.famousbirthdays.com>

WordNet (e.g. “*rabbit*”/ “*composer*”). All pairs of proper names are candidates. We ran LEILA on the Wikigeography corpus, because this set is particularly rich in synonyms. LEILA performed reasonably well. The patterns found include “*X was known as Y*” as well as several non-grammatical constructions such as “*X (formerly Y)*”.

Type. For the type relation, it is difficult to select example pairs, because if an entity belongs to a concept, it also belongs to all super-concepts. However, admitting each pair of an entity and one of its super-concepts as an example would result in far too many false positives. The problem is to determine for each entity the (super-)concept that is most likely to be used in a natural language definition of that entity. Psychological evidence [114] suggests that humans prefer a certain layer of concepts in the taxonomy to classify entities. The set of these concepts is called the *Basic Level*. Heuristically, we found that the lowest super-concept in WordNet that is not a compound word is a good approximation of the basic level concept for a given entity. We used all pairs of a proper name and the corresponding basic level concept of WordNet as examples. We could not use pairs of proper names and incorrect super-concepts as counterexamples, because our corpus Wikipedia knows more meanings of proper names than WordNet. Therefore, we used all pairs of a common noun and an incorrect super-concept from WordNet as counterexamples. All pairs of a proper name and a WordNet concept are candidates.

The Type Relation on Different Corpora. We ran LEILA on the Wikicomposers corpus. The performance on this task was acceptable, but not impressive. However, the chances to obtain a high recall and a high precision were significantly decreased by our tough evaluation policy: The ideal pairs include tuples deduced by resolving syntactic and semantic ambiguities and anaphoras. Furthermore, our evaluation policy demands that non-defining words such as *member* not be chosen as instance concepts. In fact, a high proportion of the incorrect assignments were *friend*, *member*, *successor* and *predecessor*, decreasing the precision of LEILA. Thus, compared to the gold standard of humans, the performance of LEILA can be considered reasonably good. The patterns found include the Hearst patterns [68] “*Y such as X*”, but also more complex patterns such as “*X was known as a Y*”, “*X [...] as Y*”, “*X [...] can be regarded as Y*” and “*X is unusual among Y*”. Some of these patterns could not have been found by primitive regular expression matching.

To test whether thematic heterogeneity influences LEILA, we ran it on the Wikigeneral corpus. Finally, to try the limits of our system, we ran it on the Googlecomposers corpus. As shown in Table 8, the performance of LEILA dropped in these increasingly challenging tasks, but LEILA could still produce useful results. We can improve the results on the Googlecomposers corpus by adding the Wikicomposers corpus for training.

The different learning methods (kNN and SVM) performed similarly for all relations. Of course, in each of the cases, it is possible to achieve a higher precision at the price of a lower recall. The runtime of the system splits into parsing ($\approx 40s$ for each document, e.g., 3:45h for Wikigeography) and the core algorithm (2-15min for each corpus, 5h for the huge Wikigeneral).

4.3.2.2 Results with different competitors

TextToOnto. We compared LEILA to TextToOnto and Text2Onto for the *type* relation on the Wikicomposers corpus. TextToOnto requires an ontology as source of possible concepts. We gave it the WordNet ontology, so that it had the same preconditions as LEILA. Text2Onto does not require any input. Table 9 shows the results of the comparison. Text2Onto seems to have a precision comparable to ours, although the small number of found pairs does not allow a significant conclusion. Both systems have drastically lower recall than LEILA – even though both systems are tailored to and hence restricted to the *type* relation.

Table 9: Results on Wikicomposers with the Type Relation

System	#D	#O	#C	#I	Precision	Recall	F1
LEILA(SVM)	87	685	408	1127	59.56%	36.20%	45.03%
LEILA(kNN)	87	790	463	1127	58.61%	41.08%	48.30%
Text2Onto	87	36	18	1127	50.00%	1.60%	3.10%
TextToOnto	87	121	47	1127	38.84%	4.17%	7.53%

Abbreviations as in Table 8

Again, all confidence intervals for precision and recall were below $\pm 10\%$ (see [133] for the values).

Snowball. We compared LEILA and Snowball on the original Snowball corpus with the *headquarters* relation. Unfortunately, we only had access to the test corpus. Hence we trained LEILA on a small portion (3%) of the test documents and tested on the remaining ones. Since the original 5 seed pairs that Snowball used did not appear in the collection at our disposal, we chose 5 other pairs as examples. We used no counterexamples and hence omitted the Training Phase of our algorithm. LEILA quickly finds the pattern “*Y-based X*”. This led to very high precision and good recall, compared to Snowball – even though Snowball was trained on a much larger training collection. Table 10 shows the results.

Table 10: Results on the Snowball Corpus with Headquarters

M	System	#D	#O	#C	#I	Precision	Recall	F1
S	LEILA(SVM)	54	92	82	165	89.13%	49.70%	63.81%
S	LEILA(kNN)	54	91	82	165	90.11%	49.70%	64.06%
S	Snowball	54	144	49	165	34.03%	29.70%	31.72%
I	LEILA(SVM)	54	50	48	126	96.00%	38.10%	54.55%
I	LEILA(kNN)	54	49	48	126	97.96%	38.10%	54.86%
I	Snowball	54	64	31	126	48.44%	24.60%	32.63%

M – Metric (S: Standard, I: Ideal). Other abbreviations as in Table 8

CV System. The CV-system differs from LEILA, because its ideal pairs are a table, in which each entity is assigned to its most likely concept according to a human understanding of the text – independently of whether there are explicit definitions for the entity in the text or not. We conducted two experiments:

First, we used the document set used in Cimiano and Völker’s original paper [43], the Lonely Planet corpus. To ensure a fair comparison, we trained LEILA separately on the Wikicomposers corpus, so that LEILA cannot have example pairs in its output. The relation is the *type* relation. For the evaluation, we calculated precision and recall with respect to an ideal table provided by the authors. Since the CV-system uses a different ontology, we allowed a distance of 4 edges in the WordNet hierarchy to count as a match (for both systems). Since the explicit definitions that our system relies on were sparse in the corpus, LEILA performed worse than the competitor. Table 11 shows the results (under the relaxed ideal metric).

Table 11: Results on Lonely Planet Corpus with Type Relation

System	#D	#O	#C	#I	Precision	Recall	F1
LEILA(SVM)	–	159	42	289	26.42%	14.53%	18.75%
LEILA(kNN)	–	168	44	289	26.19%	15.22%	19.26%
CV-system	–	289	92	289	31.83%	31.83%	31.83%

Abbreviations as in Table 8

In a second experiment, we had the CV-system run on the Wikicomposers corpus. As the CV-system requires a set of target concepts, we gave it the set of all concepts in our ideal pairs. Furthermore, the system requires an ontology on these concepts. We gave it the WordNet ontology, pruned to the target concepts with their super-concepts. We evaluated by the Relaxed Ideal Metric, again allowing a distance of 4 edges in the WordNet hierarchy to count as a match (for both systems). This time, our competitor performed worse (see Table 12). This is because our ideal table is constructed from the definitions in the text, which our competitor is not designed to follow.

Table 12: Results on Wikicomposers Corpus with Type Relation

System	#D	#O	#C	#I	Precision	Recall	F1
LEILA(SVM)	87	336	257	744	76.49%	34.54%	47.59%
LEILA(kNN)	87	367	276	744	75.20%	37.10%	49.68%
CV-system	87	134	30	744	22.39%	4.03%	6.83%

Abbreviations as in Table 8

These experiments only serve to show the different philosophies in the definition of the ideal pairs for the CV-system and LEILA. The CV-system does not depend on explicit definitions, but it is restricted to the *type*-relation.

4.4 Conclusion

Summary. This section has introduced the information extraction tool LEILA. A linguistic analysis of the input sentences allows LEILA to learn patterns that are robust against surface variations. By taking into account counterexamples, LEILA can also identify negative patterns. LEILA uses machine learning to generalize positive and negative patterns, thus becoming robust against false positive patterns. Our theoretical analysis has shown that the influence of false

positive patterns converges to zero as the corpus size grows. Our experiments have shown that LEILA outperforms other state-of-the-art methods in information extraction.

Discussion. LEILA uses a deep linguistic analysis of the input sentence. Thus, its patterns are more robust to surface variations than the patterns employed by other systems, such as Snowball [2]. This observation is likely to carry over to systems such as KnowItAll [56] or Alice [11]. These systems also use non-linguistic patterns, but were not considered in the experiments because they use the whole corpus as a Web. If the Web is used as a corpus, the situation changes: Linguistic parsing is barely applicable, due to the time needed to parse a single document. On the other hand, stable linguistic patterns are also of lesser importance in this scenario, because the system can rely on the redundancy of the Web. This tradeoff entails that non-linguistic systems may be the better choice for use on the Web, whereas LEILA is more adequate for a scenario with a given corpus of documents.

Unfortunately, LEILA cannot produce ontological facts directly. First, LEILA only extracts words, not entities. Thus, a disambiguation process would be necessary to make LEILA extract ontological facts in a canonical form. LEILA shares this problem with all approaches discussed in this chapter. Second, LEILA requires counterexamples for the target relation. These may be difficult to provide for certain relations. Third, LEILA shows weak performance when applied to inhomogeneous texts (such as random Internet documents). In the next chapter, we will show how these issues can be overcome. LEILA will be adapted for use with YAGO. For this purpose, the linguistic analysis of LEILA will have to be simplified. The idea of using counterexamples for pattern learning, however, will remain. It is the crucial ingredient of our integrative knowledge gathering system SOFIE.

Chapter 5

SOFIE

The goal of this thesis is to describe a system that can gather ontological knowledge. This chapter introduces the third component of this system, the information integration system SOFIE¹. SOFIE integrates the information extracted by LEILA into YAGO, thus forming the synthesis of the first two components and completing the system. This chapter will first give an overview of SOFIE and related work. It will then proceed to describing the model of SOFIE and its implementation. The last section will present an empirical validation of our approach.

5.1 Overview

5.1.1 Problem statement

This chapter discusses a very general task: the automatic gathering of ontological knowledge². Gathering knowledge for an ontology is different from pure information extraction as discussed in Chapter 4. On one hand, it is more difficult, because words have to be disambiguated to entities. On the other hand, it can be easier, because the information that is already present in the ontology can help improving the extraction quality. Consider an example. Assume that a knowledge gathering system encounters the following sentence

Einstein attended secondary school in Germany.

Knowing that “*Einstein*” is the family name of Albert Einstein and knowing that Albert Einstein was born in Germany, the system might deduce that “*X attended secondary school in Y*” is a good indicator of *X* being born in *Y*. Now imagine the system finds the sentence

Elvis attended secondary school in Memphis.

Many people have called themselves “*Elvis*”. But in the present case, assume that the context indicates that Elvis Presley is meant. But the system already

¹Self-Organizing Framework for Information Extraction

²This task is sometimes split into *Ontology Learning* (the gathering of new relations and classes) and *Ontology Population* (the extraction of new individuals and relation instances) [91, 40].

knows (from the facts it has already gathered) that Elvis Presley was born in the State of Mississippi. Knowing that a person can only be born in a single location and knowing that Memphis is not located in Mississippi, the system concludes that the pattern “*X attended secondary school in Y*” cannot mean that *X* was born in *Y*. Re-considering the first sentence, it finds that “*Einstein*” could have meant Hermann Einstein instead. Hermann was the father of Albert Einstein. Knowing that Hermann went to school in Germany, the system figures out that the pattern “*X attended secondary school in Y*” rather indicates that someone was educated in some place. This, in turn, makes it deduce that Elvis was educated in Memphis³.

It is this kind of interaction between the ontology and the information extraction that is needed to deduce ontological facts of high quality. Here, we aim at a system that can accomplish this bridging in a unified framework.

5.1.2 Related Work

There are numerous approaches that aim at gathering ontological facts. Since the task is so general, the related work spans a wide area, including the related work discussed already for YAGO (Section 3.1.2) and for LEILA (Section 4.1.2). This section aims at discussing all related work, pointing to the respective previous discussions whenever possible.

Non-Ontological Approaches. Some systems extract non-ontological information. This means, for example, that their output does not have the form of facts, that they do not disambiguate words to entities or that they do not extract well-defined relations. These approaches are presented in more detail in Section 4.1.2 ([2, 124, 105, 45, 65, 61, 162, 111, 22, 125, 154, 105]) and Section 3.1.2 ([56, 28, 10, 55]). LEILA also belongs to the non-ontological approaches. In this chapter, we aim at a system that produces ontological knowledge.

Related Approaches with Different Goals. Some approaches aim at discovering new relations ([90, 144]). This task can be seen as a sub-discipline of ontological knowledge gathering. Other approaches aim at classifying patterns into relation clusters ([37, 160]), a remotely related goal, which is explained in more detail in Section 4.1.2. The DBLife project [52] uses information extraction, but aims at building a community portal rather than an ontology. The LIBRA project [161, 100] has a slightly different focus, aiming at discovering entities in Web pages. Here, we aim at extending an ontology.

Focused Approaches. Some approaches aim at extracting instances of a single relation, mostly the *type* relation. These include [43, 64, 41, 123, 42] and also some approaches discussed in more detail in Section 4.1.2 ([30, 60, 155, 42, 25, 41, 45]). Here, we aim at a system that can add knowledge about any relation. Some approaches are designed for a certain area, such as the biomedical domain ([109]) or the business domain ([117]). Here, we aim at a general purpose approach.

³This is actually true. Albert Einstein went to secondary school in Switzerland.

Semi-Structured Approaches on Wikipedia. A number of approaches extract ontological information from the semi-structured part of Wikipedia ([8, 107, 145]). These approaches are presented in detail in Section 3.1.2. YAGO belongs to this class as well. Here, we aim at gathering even more knowledge, by exploiting also the natural language part of Wikipedia and other resources.

Human Expert Approaches. There exist projects for constructing ontological knowledge by hand ([59, 95, 102]). These approaches are presented in more detail in Section 3.1.2. The results are of impressive quality, but require costly human work. This holds also true for the semi-automatic approaches to Ontology Learning ([113, 92, 4, 91, 32]). Again other approaches that require human input are presented in more detail in Section 4.1.2 ([75, 126, 155, 153]). Here, we aim at an automated system.

Ontology-Internal Approaches. Some approaches aim at finding a mapping between ontologies ([108]) or at deriving new knowledge from within an ontology ([47]). One prominent approach in this direction is the Linking Open Data project [18], which is described in more detail in Section 3.1.2. In this chapter, we aim at a system that can find information that has not yet been codified into ontologies.

Community-Based Approaches. Some projects are based on the contribution of volunteers. The most prominent approaches are the Semantic Wikipedia project [87] and Freebase⁴. As discussed in more detail in Section 3.1.2, the benefit of these approaches will depend on their acceptance by the community and effective means of quality control. The project “Intelligence in Wikipedia” [146] is an integrative approach. It builds on Wikipedia as a community resource in combination with automatic approaches for information extraction. The automatic approaches (KYLIN [150], TextRunner [10] and KOG [151]) are discussed in their respective categories in the present section.

Co-Occurrence Analysis. De Boer et al. [20] present an approach that is based on co-occurrences of entities in Web documents. The approach requires a core ontology that provides the following items:

1. a target relation (e.g., *artistGenre*)
2. a limited number of known instances of the target relation (the *seed pairs*)
3. the domain and range of the target relation (e.g., *artist* and *genre*)
4. the set of instances of the domain and of the range of the target relation

The goal of the system is to find new instances of the target relation. For this purpose, the system first chooses one instance of the range of the target relation (e.g. *Expressionism* for *Genre*). The system uses a search engine to retrieve Web documents about this instance. In each retrieved document, it finds the occurrences of instances of the domain of the target relation (i.e., for *artistGenre*, the system finds all artists). It analyzes what proportion of them

⁴<http://www.freebase.com>

also appear in the seed pairs (e.g., what proportion of the artists mentioned in the document are known as expressionistic artists in the seed pairs). Documents with a high proportion are likely to be lists of good candidates. For example, if a document mentions several artists and 70% of them are expressionistic artists, it is likely that the other artists are also expressionistic. Finally, the system constructs facts from the promising entities (such as (*VanGogh*, *artistGenre*, *Expressionism*)). The authors have run various experiments with their system. They have measured precision and recall with respect to a gold standard and achieved F-values of up to 72%.

The beauty of this approach is that it works without any pattern matching. Thus, it is applicable to both natural language text and semi-structured content alike. However, the approach relies on two assumptions: First, it requires that the target relation is not a one-to-one relation. Thus, it can be applied less well to relations such as *hasBudget*, where one element of the range is related to relatively few elements of the domain. Second, it relies on the existence of lists (such as documents listing all expressionistic authors). While these lists may exist for some relations, they may not exist for others. For example, there might be few Web documents listing all people born on a specific day in a specific year. Worse, there may be cases in which a fact occurs only once on the Web (e.g., in Wikipedia). Thus, while the approach could clearly be highly productive (e.g., in combination with YAGO), we will investigate more techniques for knowledge gathering in this chapter.

Alice. Banko et al. have presented an approach called *Lifelong Learning* [11] and implemented it in the ALICE system. ALICE is based on a core ontology (presently WordNet [59]) and aims to extend it by new facts. ALICE uses the TextRunner [10] to retrieve non-canonical facts. For example, TextRunner might gather the fact (“*bananas*”, “*provide*”, “*vitamin c*”). ALICE maps the words to entities in the core ontology (“*bananas*” to *banana* and “*vitamin c*” to *VitaminC*), presently by stemming the word and choosing its most frequent meaning (see Section 3.2.1). In addition to discovering new relations and to gathering new facts, ALICE exhibits three novel capabilities:

1. **Class Discovery:** ALICE can take known classes (such as *food*) and extend them by new classes (such as *healthyFood*).
2. **Range and Domain Discovery:** ALICE gathers relation instances and determines the domain and range of relations (such as (*fruit*, *provides*, *vitamin*)). Although this does not canonicalize the relations (they are still natural language words), it gives them a formal link to the ontology.
3. **Guided Search:** ALICE will choose new areas to explore based on the knowledge it already has. This is the implementation of the Lifelong Learning Paradigm.

The authors have run ALICE and have manually evaluated its output. Under a very strict evaluation metric, Alice achieves a precision of 78%.

ALICE has so far only been tried out on classes (such as *fruit* and *food*), and not on individuals (such as *ElvisPresley*). Furthermore, ALICE does not yet make use of logical entailments, which would be necessary to cope with our sample scenario introduced above. Thus, while the contributions of Alice are a clear step ahead, we will continue our survey of knowledge gathering techniques.

PORE. Wang et al [143] have presented an approach called *Positive-Only Relation Extraction* (PORE). PORE is a pattern matching approach, which has been implemented for relation instance extraction from Wikipedia (see Section 4.1.2 for a discussion of pattern matching). PORE is given a target relation (e.g. *directedBy* for movies) and proceeds in five steps:

1. **Seed Pair Extraction:** PORE requires a certain number of known instances of the target relation, the *seed pairs*. In the present implementation, these seed pairs are extracted automatically from the Wikipedia infoboxes (see Section 3.1.4.2 for a description of Wikipedia infoboxes).
2. **Entity Extraction:** PORE identifies all entities in Wikipedia and extracts features for them. For example, the entity *AnnieHall* is extracted from the Wikipedia article about the movie Annie Hall. Its features are (1) its class, as mentioned in the first defining sentence of the article (*comedy-Film*) and (2) the information about the entity as given in the infoboxes (*writtenBy WoodyAllen*) and (3) its categories (*romanticComedyFilm*).
3. **Pattern Extraction:** For each pair of entities that co-occur in a sentence, PORE extracts a pattern. This pattern consists of the words around the first entity, the words between the entities and the words around the second entity. The same pattern can occur with multiple entity pairs.
4. **Data Filtering:** PORE tries to determine entity pairs that are highly unlikely to be instances of the target relation. It does this by comparing the features of a candidate entity to the features of the entities in the seed pairs. For example, in most seed pairs for *directedBy*, the first entity will exhibit the feature *movie*. If in a given candidate pair, the first entity does not exhibit this feature, the pair is less likely to be an instance of *directedBy*. This step has similarities with the type checking done in YAGO (Section 3.2.2), although it is done by statistical means rather than logical ones.
5. **Candidate Classification:** PORE has some positive entity pairs (from the seed pairs) and numerous candidate pairs. It aims to classify the candidate pairs into instances of the target relation or non-instances, given their features and given the patterns in which they occur. The authors have developed an extension of a transductive classification algorithm, B-POL, which can classify the pairs given only positive examples. The pairs classified as positive are proposed as new instances of the target relation.

The authors have run PORE for 3 relations on Wikipedia. They have evaluated the results manually and achieve F-values of up to 80%.

PORE is a holistic approach, which makes clever use of entity features to figure out whether a pair of entities is an instance of the target relation or not. One aspect that appears unexplored is whether the pairs excluded by the Data Filtering could rather be used as negative samples for the classification (as it is done in LEILA, see Section 4.2.2). Furthermore, PORE does not incorporate world knowledge, which would be necessary for our sample scenario. Hence, we continue our survey of knowledge gathering methods.

KYLIN. The KYLIN system [150] forms the basis of the KOG [151] ontology. KYLIN itself aims at filling the infoboxes in Wikipedia. As the authors explain, Wikipedia’s infoboxes suffer from several shortcomings. These include incompleteness (not all articles have an infobox, not all infoboxes have all attributes), inconsistency, schema drift (i.e., multiple attributes for the same relation) and lack of typing (the domain and range of the attributes are not explicit). KYLIN embarks to ease some of these shortcomings in three steps:

1. **Article Classification:** KYLIN predicts which articles could potentially lack a given type of infobox. For example, if the majority of articles about US counties have an infobox labeled “*US County*”, but some articles about US counties lack such an infobox, then these articles should probably be equipped with a “*US County*” infobox. Currently, KYLIN uses a simple keyword-based heuristic to detect these articles. The task is closely related to determining the class of an article entity (see Section 3.2.1).
2. **Sentence Classification:** KYLIN knows all attributes that appear in the infoboxes of a given type (e.g., it knows that “*area*” is an attribute of the US county infoboxes). In the articles that do have a value for a given attribute, KYLIN attempts to find that value in the article text. For example, if “*area = 18000 sq km*” appears in the infobox of a US county article, KYLIN attempts to find the word “*18000 sq km*” in the article. This way, it collects sentences that typically express the relation of the attribute (e.g., “*has an area of X*”). By collecting these sentences as positive samples and all other sentences as negative samples, KYLIN learns a classifier on sentences for the attribute. This procedure is very similar to pattern matching (see Section 4.1.4), but since one entity is given by the title of the article, the task that KYLIN faces is different. If an infobox in another article lacks the attribute, KYLIN uses the classifier to classify all sentences in that article. If a sentence is classified as positive, KYLIN proposes the value found in the sentence as a value for the missing attribute.
3. **Link Generation:** In Wikipedia, some entity occurrences are hyperlinked to their articles (e.g., an occurrence of Albert Einstein may be a hyperlink to the Wikipedia article about Einstein). Some occurrences, however, are not linked. KYLIN spots unlinked entity occurrences and collects potential articles to which the entity could be linked. KYLIN uses heuristics as well as statistical disambiguation techniques to determine the correct article.

The authors have conducted a manual evaluation of KYLIN and report precision rates between 70% and 100%.

KOG. KOG [151] is the KYLIN Ontology Generator. KOG first uses KYLIN to complete and add infoboxes on Wikipedia. Then, it sets out to clean the infobox data in 4 steps:

1. **Recognizing Duplicate Infobox Types:** Sometimes, the same class of things (e.g., US counties) are described by two types of infoboxes (e.g., *uscounty* and *us.county*). KOG uses the Wikipedia redirect pages, the

Wikipedia editing history and name similarity heuristics to detect and merge equivalent infobox types.

2. **Assigning Meaningful Names:** Some infobox types have obscure names such as *abl*. KOL uses the Wikipedia redirect links, the Wikipedia categories of the respective articles and Google spell checking to find more meaningful names for the types (such as *AmericanBaseballLeague* instead of *abl*).
3. **Inferring Attribute Ranges:** KOG collects all values of a given attribute (e.g., all people who occur as values of the attribute *directedBy*). For each value, it determines the class of which the value is an instance. KOG uses the YAGO hierarchy and the DBpedia hierarchy [8] for this purpose. It chooses the superclass of the most frequent classes as the range for the attribute. For example, if most people that appear in a *directedBy* attribute are instances of the class *AmericanMovieDirectors* or the class *ItalianMovieDirectors*, KOG chooses the superclass *movieDirectors*. This task is similar to the task solved by ALICE [11], see above.
4. **Attribute Mapping:** Some attributes are similar, even if they appear in different infobox types. For example, both the infoboxes for actors and the infoboxes for scientists have an attribute *spouse*. KOG uses string matching techniques, information from the taxonomy (s.b.) and data from the edit history of pages to identify such attributes.

Next, KYLIN builds a taxonomy in 2 phases:

1. **Article Subsumption:** Using name matching, pattern matching and information from the Wikipedia categories, the edit histories, and WordNet [59], KOG establishes which article entity must be a subclass of which other article entity. For example, KOG figures out that *EnglishPublicSchool* must be a subclass of *PublicSchool*.
2. **Connection to WordNet:** As YAGO, KOG aims to connect the Wikipedia entities to WordNet. For this purpose, KOG uses the mapping that exists already in YAGO and the mapping that exists already in DBpedia [8], and trains a classifier to detect subsumption between a Wikipedia entity and a WordNet entity. KOG uses Markov Logic Networks [112] to take into account rules such as the transitivity of *subClassOf*.

The authors have evaluated the output of KOG and report very high precision rates. In particular, the mapping to WordNet has an accuracy of 97% (just as YAGO's). KOG uses ideas from YAGO (joining Wikipedia and WordNet) and from DBpedia (taking attributes as relations). It improves on YAGO by automatically discovering new relations. It improves on DBpedia by canonicalizing the relations and by going beyond the data given in the infoboxes. KOG is very successful, but tailored to Wikipedia. We aim at discovering an approach that can deal with arbitrary input.

Declarative Information Extraction. Shen et al. [120] propose a framework called *declarative information extraction*. The building blocks of this framework are *procedural predicates*. A procedural predicate is a small piece

of side-effect-free programming code (e.g., a function or method in C or Java). Such a predicate acts as a function, which, given some arguments as input, runs an algorithm and either (1) fails or (2) produces some arguments as output. For example, one procedural predicate could decide whether a certain substring appears in another string. The framework of declarative information extraction allows combining the procedural predicates to *rules* (in the form of Datalog). For example, one rule can state that X must be the author of paper Y if X appears in at least three documents together with the word “*author*” and paper Y . Finally, automated reasoning on the rules can be used to extract information from documents. The authors show that their approach allows for optimization techniques that can speed up the information extraction considerably. By encapsulating the non-declarative code into procedural predicates, the framework provides a clean, declarative model for rule-based information extraction. The rules of the system, however, have to be designed manually. It would be desirable if human input could be reduced even further.

Logical Knowledge. All of the approaches mentioned here have their benefits and have advanced the understanding of information extraction and ontology construction. We observe, however, that none of the approaches takes into account logical background knowledge – such as for example that people must be born before they die. Such knowledge might be highly useful, because it can exclude impossible hypotheses and deduce safe hypotheses. As shown in our sample scenario, there may be cases where logical reasoning is mandatory. Although several authors have mentioned the possibility of logical reasoning ([120, 151, 11]), it has not been implemented. It would be nice to have an approach that unifies information extraction, ontology construction and logical reasoning. This is what SOFIE does.

5.1.3 Contribution

This chapter presents SOFIE, a new system for ontological knowledge gathering. SOFIE casts the task of information extraction into a logical reasoning problem. In SOFIE, word disambiguation, pattern matching, and rule-based reasoning on the ontology all become part of one unified framework. More precisely, SOFIE combines three capabilities in a single system:

1. **Word Disambiguation:** SOFIE can take into account all meanings of a word. Based on disambiguation strategies, it can decide for the most likely meaning of a word. Different from existing systems, however, SOFIE will automatically re-consider the disambiguation if more evidence for another meaning becomes apparent.
2. **Pattern Matching:** Like LEILA and many other systems (see Section 4.1.2), SOFIE finds patterns in text documents to extract facts. Unlike the other systems, SOFIE can reason on the plausibility of patterns and reject patterns if counter evidence becomes available.
3. **Ontological Reasoning:** SOFIE makes full use of the background knowledge of the ontology. SOFIE can take into account constraints on the relations, links between hypotheses and connections to the existing knowledge.

All of these components are naturally joined in the unifying framework of SOFIE. The goal of SOFIE is to *understand* a text in the sense of Section 1.3.1. The following sections will first introduce the model of SOFIE. Then, they will discuss the implementation of SOFIE and present our experiments.

5.2 Model

SOFIE is designed to extend an existing ontology. Hence, in the following, we assume a given ontology. For our experiments, we used YAGO, but our approach is open to any ontology. SOFIE extracts new information from text documents. Hence, in the following, we assume a given corpus of documents. In our experiments, we used documents downloaded from the Internet, but any other source can be imagined. SOFIE casts the information extraction into a logical reasoning problem. We will first introduce the notion of *statements* and then proceed to the notion of *rules*. Finally, we will show how the model can be cast into a weighted MAX SAT problem.

5.2.1 Statements

Wics. As a knowledge gathering system, SOFIE has to address the problem of polysemy. In general, most words have several meanings. The word “Java”, for example, can refer to the programming language or to the Indonesian island. In a given context, however, a word is very likely to have only one meaning [62]. For example, the occurrences of the word “Java” in one Web document are likely to refer either all to the island or all to the programming language (unless the document is about the problems of polysemy). This gives rise to the following definition:

DEFINITION 18: [*Word in Context*]

A *word in context* (wic) is a pair of a word and a context.

For us, the context of a word will simply be the document in which the word appears. Thus, a wic is essentially a pair of a word and a document identifier⁵. We use the notation *word@doc*. For example, we identify the word “Java” in the document *D8* by

Java@D8

This way, the word “Java” in the document about Indonesian islands forms one wic, whereas the word “Java” in another document (be it about islands or programming languages) forms another wic. Once one occurrence of a wic has been disambiguated, it is assumed that all other occurrences of the wic are disambiguated as well. For example, once it has been figured out that document *D8* is about programming languages, all occurrences of *Java@D8* will be assumed to refer to the programming language. When talking about patterns in text documents, we will henceforth be precise and say that a pattern appears

⁵A wic is related to a *KWIC* (keyword in context), also known as a *concordance*. A concordance is a word together with an ordered vector of the immediate surrounding words [94]. Thus, two occurrences of the same word in one document may form different concordances, but only one wic.

with two *wics* instead of two *words*. Following the all-embracing definition of entities (Section 1.3.2), wics are also entities.

Facts and Hypotheses. For SOFIE, we will deal with relations of arbitrary arity. Hence we use a prefix notation for statements. Each statement can have an associated *truth value* of 1 or 0. We denote the truth value of a statement in square brackets:

bornIn(*AlbertEinstein*, *Ulm*)[1]

In compliance with the definition in Section 1.3.2, a statement with truth value 1 is called a *fact*. A statement with an unknown truth value is called a *hypothesis*. We will now see how both the ontology and the corpus can be interpreted as sources of facts.

Ontological Facts. SOFIE is designed to extend an existing ontology. We consider the ontology a set of facts. In the case of YAGO, this looks as follows: Technically speaking, the YAGO ontology is a reification graph. With our definitions, however, YAGO can also be interpreted as a set of facts. Here is an excerpt of YAGO in this light:

bornIn(*AlbertEinstein*, *Ulm*)[1]
bornOnDate(*AlbertEinstein*, *1879-03-14*)[1]
 ...

This representation can also be enhanced by including the fact identifier of each fact as an additional argument. This would allow representing the *n*-ary relations of YAGO (see Section 2.2.3). For simplicity, however, we limit ourselves to the binary facts in YAGO for the time being.

Textual Facts. LEILA will extract textual information from the corpus. This information also takes the form of facts. One type of facts makes assertions about the number of times that a pattern occurred with two wics. For example, LEILA might find that the pattern “*X went to school in Y*” occurred with the wics *Einstein@D29* and *Germany@D29*:

patternOcc(“*X went to school in Y*”, *Einstein@D29*, *Germany@D29*)[1]

Another type of facts can state how likely it is from a linguistic point of view that a wic refers to a certain entity. We call this likeliness value the *disambiguation prior*. We will discuss later how the disambiguation prior can be computed. Here, we just give an example for facts about the disambiguation prior of the wic *Elvis@D29*:

disambPrior(*Elvis@D29*, *ElvisPresley*, 0.8)[1]
disambPrior(*Elvis@D29*, *ElvisCostello*, 0.2)[1]

Other types of textual facts can be imagined. For example, LEILA could produce facts that tell which wic occurred in which document or which wic occurred how often with which other wic.

Hypotheses. Based on the ontological facts and the textual facts, SOFIE will form hypotheses. These hypotheses can concern the disambiguation of wics.

For example, SOFIE can hypothesize that *Java@D8* should be disambiguated as the programming language Java:

$$\text{disambiguateAs}(\text{Java@D8}, \text{JavaProgrammingLanguage})[?]$$

We use a question mark to indicate the unknown truth value of the statement. SOFIE will also hypothesize about whether a certain pattern expresses a certain relation:

$$\text{expresses}(\text{"X was born in Y"}, \text{bornInLocation})[?]$$

SOFIE also forms hypotheses about potential new facts. For example, SOFIE could establish the hypothesis that Java was developed by Microsoft:

$$\text{developed}(\text{Microsoft}, \text{JavaProgrammingLanguage})[?]$$

Unification. By casting both the ontology and the corpus analysis into statements, SOFIE unifies the domains of ontology and information extraction. For SOFIE, there exist only statements. SOFIE will try to figure out which hypotheses are likely to be true. For this purpose, SOFIE uses *rules*.

5.2.2 Rules

Literals and Rules. SOFIE will use logical background knowledge to figure out which hypotheses are likely to be true. This knowledge takes the form of *rules*. Rules are based on *literals*:

DEFINITION 19: [*Literal*]

A *literal* is a statement that can have placeholders for the relation or some of the entities.

Here is an example of a literal with uppercase strings as placeholders:

$$\text{bornIn}(X, \text{Ulm})$$

Now, a rule is basically a propositional formula over literals:

DEFINITION 20: [*Rule*]

A *rule* over a set of literals \mathcal{L} is one of the following

- an element of \mathcal{L}
- an expression of the form $\neg R$, where R is a rule over \mathcal{L}
- an expression of the form $(R_1 \diamond R_2)$, where R_1 and R_2 are rules over \mathcal{L} and $\diamond \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

As usual, we omit the outermost brackets and brackets around \wedge -expressions. With these conventions, the following line is a rule stating that whoever is born in Ulm is not born in Timbuktu:

$$\text{bornIn}(X, \text{Ulm}) \Rightarrow \neg \text{bornIn}(X, \text{Timbuktu})$$

As in Prolog and Datalog, all placeholders are implicitly universally quantified. We postpone the discussion of the formal semantics of the rules to Section 5.2.3 and stay with an intuitive understanding of rules for the moment.

Grounding. The relation between statements and literals is as follows:

DEFINITION 21: [*Ground Instances*]

A *ground instance of a literal* is a statement obtained by replacing the placeholders by entities. A *ground instance of a rule* is a rule obtained by replacing all placeholders by entities. All occurrences of one placeholder must be replaced by the same entity.

For example, the following is a ground instance of the rule mentioned above:

$$\text{bornIn}(\text{AlbertEinstein}, \text{Ulm}) \Rightarrow \neg \text{bornIn}(\text{AlbertEinstein}, \text{Timbuktu})$$

SOFIE's Rules. We have developed a number of rules for SOFIE. One of the rules states that a functional relation should not have more than one second argument for a given first argument:

$$\begin{array}{l} R(X, Y) \\ \wedge \quad \text{type}(R, \text{function}) \\ \wedge \quad \text{different}(Y, Z) \\ \Rightarrow \quad \neg R(X, Z) \end{array}$$

This rule uses the virtual relation *different* (see Section 2.2.8). The rule guarantees, for example, that people are not born in more than one place. Since *disambiguatedAs* is also a functional relation, the rule also guarantees that one wic is disambiguated to at most one entity. In some sense, this rule is already employed during the construction of YAGO (see Section 3.2.2).

There are also other rules, some of which concern the textual facts. One rule says that if pattern P occurs with entities x and y and if there is a relation r , such that $r(x, y)$, then P expresses r . For example, if the pattern “ X was born in Y ” appears with Albert Einstein and his true location of birth, Ulm, then it is likely that “ X was born in Y ” expresses the relation *bornInLocation*. A naive formulation of this rule looks as follows:

$$\begin{array}{l} \text{patternOcc}(P, X, Y) \\ \wedge \quad R(X, Y) \\ \Rightarrow \quad \text{expresses}(P, R) \end{array}$$

We need to take into account, however, that patterns hold between wics, whereas facts hold between entities. Our model allows us to incorporate this constraint in an elegant way:

$$\begin{array}{l} \text{patternOcc}(P, WX, WY) \\ \wedge \quad \text{disambiguatedAs}(WX, X) \\ \wedge \quad \text{disambiguatedAs}(WY, Y) \\ \wedge \quad R(X, Y) \\ \Rightarrow \quad \text{expresses}(P, R) \end{array}$$

There exists also a dual version of this rule: If the pattern expresses the relation r , and the pattern occurs with two entities x and y , and x and y are of the correct types, then $r(x, y)$:

$$\begin{array}{ll}
 & \text{patternOcc}(P, WX, WY) \\
 \wedge & \text{disambiguatedAs}(WX, X) \\
 \wedge & \text{disambiguatedAs}(WY, Y) \\
 \wedge & \text{domain}(R, DOM) \\
 \wedge & \text{type}(X, DOM) \\
 \wedge & \text{range}(R, RAN) \\
 \wedge & \text{type}(Y, RAN) \\
 \wedge & \text{expresses}(P, R) \\
 \Rightarrow & R(X, Y)
 \end{array}$$

By this rule, we are making a design choice: The pattern comes into play only if the two entities are of the correct type. Thus, the very same pattern can express different relations if it appears with different types. We will see in the experiments (Section 5.4) how this works in practice. Another rule makes sure that the disambiguation prior influences the choice of disambiguation:

$$\begin{array}{ll}
 & \text{disambPrior}(W, X, N) \\
 \Rightarrow & \text{disambiguatedAs}(W, X)
 \end{array}$$

Softness. More rules can be added, for example the rules from Section 2.2.5. In general, it is impossible to satisfy all of these rules simultaneously. For example, as soon as there exist two disambiguation priors for the same wic, both will enforce a certain disambiguation. Two disambiguations, however, contradict the functional constraint of *disambiguatedAs*. This is why certain rules will have to be violated. Some rules are less important than others. For example, if a strong disambiguation prior requires a wic to be disambiguated as X , while a weaker prior desires Y , then X should be given preference – unless other constraints favor Y . This is why a sophisticated approach is needed to compute the most likely hypotheses. Before the next section discusses such a sophisticated approach, Figure 8 summarizes our notions again.

Ontological Facts	Textual Facts	Hypotheses
<code>bornIn(Einstein, Ulm)[1]</code> ...	<code>patternOcc(...)[1]</code> <code>disambPrior(...)[1]</code> ...	<code>expresses(</code> <code> “X born in Y”,</code> <code> bornIn)[?]</code> <code>developed(</code> <code> Microsoft,</code> <code> JavaProg.)[?]</code> ...
Rules		
$R(X, Y) \wedge \text{type}(R, \text{function}) \wedge \text{different}(Y, Z) \Rightarrow \neg R(X, Z)$...		

Figure 8: Statements and Rules

5.2.3 MAX-SAT Model

Design Alternatives. Abstractly speaking, SOFIE aims to find the hypotheses that should be accepted as true so that a maximum number of rules is satisfied. Different approaches are conceivable. We sketch each of them informally before embarking on a formal definition of our approach.

- The problem can be cast into a rule-based setting, for example into a Datalog program (as it is done in [120]). However, this would not allow violating certain rules – which is a necessary desideratum.
- The problem can be cast into a *maximum satisfiability problem* (*MAX SAT problem*). The MAX SAT problem is the task of, given a set of boolean variables and a set of propositional logic formulae, assigning truth values to the variables so that the number of satisfied formulae is maximized. In our setting, the variables would be the hypotheses and the rules would be transformed to propositional formulae on them. This view would allow violating some rules, but it would not allow weighting them.
- The problem can be cast into a Markov Logic Network [112]. A Markov Logic Network is concerned with a set of weighted first order logic formulae f_1, \dots, f_n over literals. For each formula f_i , the Markov Logic Network defines a function ϕ_i . ϕ_i takes as argument a possible *state of the world*, i.e., a possible assignment of truth values to the ground instances of the literals (e.g. $\{\text{bornIn}(\text{Einstein}, \text{Ulm})=\text{true}, \text{bornIn}(\text{Einstein}, \text{Timbuktu})=\text{false}, \dots\}$)⁶. ϕ_i returns a real value that grows monotonically with the number of ground instances of f_i that are satisfied by the assignment.

⁶See [112] for a more precise definition of literals, ground instances and formulae and for the necessary assumptions.

It can be shown that the product $\prod_i \phi_i$ defines a probability distribution over the possible states of the world. In particular, states that satisfy a higher number of formulae have a higher probability. Markov Logic Networks are very powerful and could be used to model our problem. However, such a model would lift the problem to a more complex level (that of inferring probabilities), usually involving heavy machinery. Furthermore, Markov Logic Networks might not be able to deal efficiently with the millions of facts that YAGO provides.

We chose a fourth option, which is simple, yet powerful enough to model our problem: the weighted maximum satisfiability setting (Weighted MAX SAT).

Weighted MAX SAT. The weighted MAX SAT problem is based on the notion of clauses:

DEFINITION 22: [*Clause*]

A clause C over a set of variables \mathcal{X} consists of

- a positive literal set $c^1 = \{x_1^1, \dots, x_n^1\} \subseteq \mathcal{X}$
- a negative literal set $c^0 = \{x_1^0, \dots, x_m^0\} \subseteq \mathcal{X}$

We denote C by

$$(x_1^1 \vee x_2^1 \vee \dots \vee x_n^1 \vee \neg x_1^0 \vee \neg x_2^0 \vee \dots \vee \neg x_m^0)$$

A *weighted clause* over \mathcal{X} is a clause C over \mathcal{X} with an associated weight $w(C) \in \mathbb{R}^+$.

Given a clause C over a set \mathcal{X} of variables, we say that a variable $x \in \mathcal{X}$ appears with *polarity* p in C , if $x \in C^p$. Consider an example: If $\mathcal{X} = \{w, x, y, z\}$ is a set of variables, then the following is a clause over \mathcal{X} :

$$(w \vee x \vee \neg y \vee \neg z)$$

In this clause, w and x appear with positive polarity and y and z appear with negative polarity. Intuitively, the clause says that one of the variables w, x should be assigned a truth value of 1, while one of the variables y, z should be assigned a truth value of 0. This intuition is formalized as follows:

DEFINITION 23: [*Assignment, Partial Assignment, Satisfying Assignment*]

An *assignment* for a set \mathcal{X} of variables is a function $v : \mathcal{X} \rightarrow \{0, 1\}$. A *partial assignment* for \mathcal{X} is a partial function $v : \mathcal{X} \rightarrow \{0, 1\}$. A (partial) assignment for \mathcal{X} *satisfies* a clause C over \mathcal{X} , if there is an $x \in \mathcal{X}$, such that $x \in C^{v(x)}$.

In the example, an assignment v with $v(w) = 1, v(x) = 1, v(y) = 0, v(z) = 1$ would be a satisfying assignment. We use the notation $\neg t = 1 - t$ for truth values t . Now we are ready to define the weighted MAX SAT problem:

DEFINITION 24: [*Weighted MAX SAT*]

Given a set \mathcal{C} of weighted clauses over a set \mathcal{X} of variables, the *weighted MAX SAT problem* is the task of finding an assignment v for \mathcal{X} that maximizes the sum of the weights of the satisfied clauses:

$$\sum_{c \in \mathcal{C} \text{ is satisfied in } v} w(c)$$

An assignment that maximizes the sum of the satisfied clauses in a weighted MAX SAT problem is called a *solution* of the problem.

SOFIE. Intuitively speaking, the problem that SOFIE faces is, given a set of facts, a set of hypotheses and a set of rules, finding truth values for the hypotheses so that a maximum number of rules is satisfied. Now, this problem can be cast into a weighted MAX-SAT problem. In the following, we assume a finite set of rules. Furthermore, we assume a finite set of ontological facts and a finite set of textual facts. These assumptions implicitly define a finite set of entities. We proceed as follows:

1. Every rule is syntactically replaced by all of its grounded instances. Since the set of entities is finite, the set of ground instances is finite as well.
2. Each ground instance is transformed to one or multiple clauses as usual in propositional logic. Here, we give a pattern that captures all rules introduced in Section 5.2.2:

$$p_1 \wedge \dots \wedge p_n \Rightarrow c \quad \rightsquigarrow \quad (\neg p_1 \vee \dots \vee \neg p_n \vee c)$$

3. The set of all statements that appear in the clauses becomes the set of variables. Note that these statements will include not only the ontological facts and the textual facts, but also all hypotheses that the rules construct from them.

These steps leave us with a set of variables and a set of clauses. Figure 9 exemplifies this process:


<i>Rule:</i>	$\text{bornIn}(X, \text{Ulm}) \Rightarrow \neg \text{bornIn}(X, \text{Timbuktu})$
<i>Ground instances:</i>	$\text{bornIn}(\text{Einstein}, \text{Ulm}) \Rightarrow \neg \text{bornIn}(\text{Einstein}, \text{Timbuktu})$ $\text{bornIn}(\text{Microsoft}, \text{Ulm}) \Rightarrow \neg \text{bornIn}(\text{Microsoft}, \text{Timbuktu})$...
<i>Clauses:</i>	$(\neg \text{bornIn}(\text{Einstein}, \text{Ulm}) \vee \neg \text{bornIn}(\text{Einstein}, \text{Timbuktu}))$ $(\neg \text{bornIn}(\text{Microsoft}, \text{Ulm}) \vee \neg \text{bornIn}(\text{Microsoft}, \text{Timbuktu}))$... 
<i>Variables:</i>	$\text{bornIn}(\text{Einstein}, \text{Ulm}), \text{bornIn}(\text{Einstein}, \text{Timbuktu}),$ $\text{bornIn}(\text{Microsoft}, \text{Ulm}), \text{bornIn}(\text{Microsoft}, \text{Timbuktu}),$...

Figure 9: Conversion to Weighted MAX SAT

Weighting. We partition the clauses into two sets as follows:

1. The clauses about the disambiguation of wics and the quality of patterns may possibly be violated. These are the clauses that contain the relation *patternOcc* or the relation *disambPrior* (see again Section 5.2.2). We

assign them a fixed weight w . For the *disambPrior* facts, we multiply w with the disambiguation prior, so that the prior analysis is reflected in the weight.

2. The other clauses should not be violated. We assign them a fixed weight W . W is chosen so large that even repeated violation (say, hundred-fold) of a clause with weight w does not sum up to the violation of a clause with weight W .

This way, every clause has a weight and we have transformed the problem into a weighted MAX SAT problem.

Ockham’s Razor. The optimal solution of the weighted MAX SAT problem shall reflect the optimal assignment of truth values to the hypotheses. In practice, however, there are often multiple optimal solutions. Among these, we prefer the solution that makes the least number of hypotheses true⁷. We encode this desideratum in our weighted MAX SAT problem by adding a clause ($\neg h$) with a small weight ε for each hypothesis h . This makes sure that no hypothesis is made true if there is no evidence for it. The exact value for ε is not relevant. Given two solutions of otherwise equal weight, ε just serves to choose the one that makes the least number of hypotheses true.

5.3 Implementation

The implementation of SOFIE is based on an adaptation of LEILA and an algorithm for the weighted MAX SAT problem. This section will discuss these two components.

5.3.1 Adaptation of LEILA

Surface Patterns. LEILA has to be adapted in order to work together with SOFIE. In the original version, LEILA produces deep linguistic patterns, in which the grammatical dependencies of the words are represented. This proved fruitful for standard natural language documents. The setting of SOFIE, however, is slightly different. First, we aim to process large numbers of documents. The Link Grammar Parser is inconvenient for this purpose, because it is the slowest component of LEILA. Second, we aim to process also information that is not in the form of natural language sentences. This includes for example the Wikipedia infoboxes as well as HTML-tables. It also concerns information in note-form (such as “*Che Guevara. Occupation: Revolutionary*”). For these reasons, we omit the linguistic analysis. For SOFIE, LEILA will work with surface patterns for the time being. It does not matter in which way the patterns are represented for the framework of SOFIE.

⁷This principle is known as *Ockham’s Razor*, after the 14th-century English logician William of Ockham. In our setting (as in reality), omitting this principle leads to random hypotheses being taken for true.

Learning. In the original version, LEILA collects patterns and uses machine learning to assess their quality. LEILA gathers patterns that appear with examples and invalidates them if they appear with counterexamples. We observe that this is exactly the effect of the rules in SOFIE: Once a pattern occurs with a (disambiguated) pair of entities that is known to be an instance of a relation, the following rule will make SOFIE assume that the pattern expresses the relation:

$$\begin{array}{lcl}
 & \text{patternOcc}(P, WX, WY) & \\
 \wedge & \text{disambiguatedAs}(WX, X) & \\
 \wedge & \text{disambiguatedAs}(WY, Y) & \\
 \wedge & R(X, Y) & \\
 \Rightarrow & \text{expresses}(P, R) &
 \end{array}$$

If the pattern occurs with some other pair, the following rule will make the pair an instance of the relation, just like LEILA would also make the pair an instance of the relation:

$$\begin{array}{lcl}
 & \text{patternOcc}(P, WX, WY) & \\
 \wedge & \text{disambiguatedAs}(WX, X) & \\
 \wedge & \text{disambiguatedAs}(WY, Y) & \\
 \wedge & \text{domain}(R, DOM) & \\
 \wedge & \text{type}(X, DOM) & \\
 \wedge & \text{range}(R, RAN) & \\
 \wedge & \text{type}(Y, RAN) & \\
 \wedge & \text{expresses}(P, R) & \\
 \Rightarrow & R(X, Y) &
 \end{array}$$

Now suppose that the pattern occurs with a pair that is definitively not an instance of the target relation (such as a wrong birth date). Then, in order to satisfy the above rule, SOFIE has two options: Either the wics have to be disambiguated differently or SOFIE has to decide that the pattern cannot express the relation. If the pattern occurs very often with counterexamples of this type, SOFIE will have to decide that the pattern cannot express the relation. Thus, the very essence of LEILA's learning algorithm is already implicitly encoded in the framework of SOFIE. Furthermore, SOFIE already possesses a huge wealth of positive example pairs for all relations by help of the existing YAGO ontology. Likewise, the rules on functional relations make sure that SOFIE also implicitly possesses an infinite number of counterexamples. This way, the actual learning phase of LEILA, including the management of examples and counterexamples, is completely absorbed in the framework of SOFIE. As a result, LEILA is reduced to a pattern generator. The only thing LEILA has to do is producing facts for pattern occurrences and disambiguation priors.

Pattern Occurrences. Being reduced to a surface pattern finding system, LEILA will take a document and produce all patterns that appear between any two entities. This procedure is shown in Algorithm 2.

ALGORITHM 2: Find Patterns**Input:** Document d Maximal pattern length k (default: 6)**Output:** Textual facts

```

1  preprocess( $d$ )
2   $t_1...t_n := \text{tokenize}(d)$ 
3   $T := \emptyset$ 
4  FOR  $i=1$  TO  $n$ 
5    IF  $t_i$  is a boundary THEN
6       $T := \emptyset$ 
7    IF  $t_i$  is an interesting token THEN
8      FOR EACH  $j \in T$ 
9        IF  $i - j > k$  THEN  $T := T \setminus \{j\}$ 
10       ELSE IF  $i - j > 0$ 
11         output  $\text{patternOcc}(\sigma(t_{j+1}) \dots \sigma(t_{i-1}), t_j@d, t_i@d)[1]$ 
12        $T := T \cup \{i\}$ 

```

This algorithm takes as input a document d and a maximal pattern length k . It first preprocesses the document by decoding all characters and normalizing all dates and numbers (line 1, as described in Section 4.2.1). Then, it *tokenizes* the document; that is, it splits the document into small strings known as *tokens*. The tokenization identifies (normalized) numbers, (normalized) dates and, in Wikipedia articles, also Wikipedia hyperlinks. Furthermore, the tokenization employs lists (such as a list of stop words, a list of nationalities and a list of US states) to identify known words. Last, the tokenization identifies strings that must be person names (e.g., because they are accompanied by a honorary title).⁸ The output of this procedure is a list of tokens (line 2). Next, the algorithm iterates through all tokens (lines 4-12) and identifies “interesting” tokens (line 5). Since we are primarily concerned with information about individuals, all numbers, dates and proper names are considered “interesting”. Whenever two interesting tokens appear within a window of maximum size k in the list of tokens, the algorithm produces a pattern (line 11). By the substitution σ , the algorithm substitutes other interesting tokens in the pattern by some fixed symbol \diamond . For example, the pattern “ X was born in 1980 in Y ” is stored as “ X was born in \diamond in Y ”. The pattern is output as a pattern occurrence fact (line 11). The algorithm registers the index of the interesting token in the set T , so that it can serve as a first argument in the next pattern occurrence (line 12). Care is taken not to cross sentence boundaries (lines 5-6). We made the experience that a maximum pattern size of 6 is a reasonable choice.

Tokenizing Wikipedia. Wikipedia is a special type of corpus, because it provides both unstructured text and structured text. To harvest the structured parts of Wikipedia as well, our tokenizer tokenizes the infoboxes and categories as follows: It inserts the article entity before each attribute name and before each category name. For example, the part “*born in = Ulm*” in the infobox about Albert Einstein is tokenized as “*Albert Einstein born in = Ulm*”. By

⁸All preprocessing tools mentioned here are available at <http://mpii.de/~suchanek/downloads/javatools>.

this minimal modification, large parts of structured text become accessible to SOFIE. Alternatively, the structured parts can be harvested by *sesquiary patterns*, explained in the following.

Sesquiary Patterns. In some cases, it is known a priori that the document is about a certain entity, the *topic* of the document. For example, every Wikipedia article is about a single topic. In this case, it is likely that most sentences in the document express a relation between the topic and some other entity. Thus, it suffices to extract unary patterns around interesting words and to complement them with the topic to binary patterns. We call such a pattern a *sesquiary pattern*⁹. Algorithm 3 illustrates this method:

ALGORITHM 3: Find Sesquiary Patterns

Input: Document d
 Maximal pattern length k (default: 3)
 Topic entity e
Output: Textual facts

```

1 preprocess( $d$ )
2  $t_1 \dots t_n := \text{tokenize}(d)$ 
4 FOR  $i=1$  TO  $n$ 
5   IF  $t_i$  is not interesting THEN CONTINUE
6    $j = i$ 
7   WHILE  $i - j < k \wedge t_j$  is no boundary DO  $j := j - 1$ 
8   output  $\text{patternOcc}(\sigma(t_{j+1}) \dots \sigma(t_{i-1}), e, t_i@d)[1]$ 
```

The algorithm takes as input a document d about an entity e and a maximal pattern size k . It first preprocesses (line 1) and tokenizes (line 2) the document (as described for Algorithm 2, without inserting the article entity). Then, it identifies all interesting tokens (line 5, again as described for Algorithm 2). For each interesting token t_i , it seeks backwards k tokens without crossing sentence boundaries (line 7). From that position, j , it constructs the pattern “ $t_{j+1} \dots t_{i-2} t_{i-1}$ ” between e and t_i (line 8). As in Algorithm 2, the substitution σ replaces interesting tokens in the pattern itself by the special symbol \diamond . The output is a set of *patternOcc* facts with sesquiary patterns.

Sesquiary patterns allow us to extract information from documents that have the form of notes. Consider as an example the following document d about John Lennon:

John Lennon
 Born in: Liverpool
 Genre: Rock
 ...

In this example, the following patterns are extracted

$\text{patternOcc}(\text{“Born :”, JohnLennon@d, Liverpool@d})[1]$
 $\text{patternOcc}(\text{“Genre :”, JohnLennon@d, Rock@d})[1]$
 ...

⁹These patterns are neither binary nor unary, but “ $1\frac{1}{2}$ ary”, that is sesquiary.

We made the experience that a maximal pattern length of 3 is usually sufficient. In general, sesquary patterns can work on information that is not in the form of proper sentences. Furthermore, sesquary patterns can be productive without anaphora resolution (see Section 4.1.2). This technique also allows taking into account the Wikipedia infoboxes in a natural way, because each line in the infobox will generate one sesquary pattern (see again Section 3.1.4.2 for the form of infoboxes in the Wiki Markup Language). Sesquary patterns can be generalized to encompass not just the preceding words, but also the following words or even other features of the surrounding text. In principle, the infobox filling algorithm of KYLIN [150] uses such a strategy when finding attribute values in the articles (see Section 5.1.2).

Disambiguation. The adapted LEILA produces pattern occurrences with wics. Each wic can have several meanings. LEILA produces a disambiguation prior for each wic. There are numerous approaches for estimating the disambiguation prior.¹⁰ For our experiments, we use a particularly simple estimation, which is known as the *bag of words approach*. It is described in Algorithm 4.

ALGORITHM 4: Compute Disambiguation Prior

Input: Wic $w@d$

Ontology o

Output: Textual facts

- 1 $context(d) :=$ set of words in d
- 2 $e_1, \dots, e_n :=$ possible meanings of w in o
- 3 **FOR** $i = 1$ **TO** n
- 4 $context(e_i) :=$ set of entities connected to e_i in o
- 5 **FOR** $i = 1$ **TO** n
- 6 $p := \frac{context(d) \cap context(e_i)}{\sum_j context(d) \cap context(e_j)}$
- 7 output $disambPrior(w@d, e_i, p)[1]$

The algorithm takes as input a wic of a word w in a document d . It also requires an ontology o . The algorithm first considers the set of words in the document d . This set is called $context(d)$ (line 1). Then, the algorithm looks up all possible meanings of w in the ontology (line 2; in YAGO this can be done by following the *means* edges from w). For each meaning e_i , the algorithm computes the set of entities connected to e_i in the ontology. This set is called $context(e_i)$ (line 4). Then, the algorithm computes the disambiguation prior of a meaning e_i as the normalized size of the intersection of $context(e_i)$ and $context(d)$ (line 6). This value increases with the amount of evidence that is present in d for the meaning e_i . By the normalization, we obtain a probability distribution over the possible meanings of w . Finally, the algorithm outputs the resulting textual facts (line 7).

We made the experience that the top 3 disambiguation priors are usually sufficient. We observe that a full disambiguation is not always necessary: First, all literals in the document are already normalized. Hence, they always refer

¹⁰See [3] for a comprehensive overview.

to themselves. Second, some words have only one meaning. For these tokens, LEILA produces no disambiguation prior. Instead, it produces pattern occurrences that contain the respective entity directly in place of the wic. The relation *disambiguateAs* is configured in such a way that each entity is mapped to itself. Algorithm 5 summarizes the complete adapted LEILA algorithm.

ALGORITHM 5: Adapted LEILA

Input: Document d
 Ontology o
 Maximal pattern size k
 Flag *sesquinary* $\in \{0, 1\}$
 Entity e if *sesquinary* = 1

Output: Textual facts

- 1 IF *sesquinary* = 0 THEN Find Patterns (Alg. 2) with d, k
- 2 ELSE Find Sesquinary Patterns (Alg. 3) with d, o, e, k
- 3 FOR EACH wic $w@d$ that has been produced
- 4 Compute Disambiguation Prior (Alg. 4) with w, d, o

5.3.2 Weighted MAX SAT Algorithm

Prior Assignments. In our weighted MAX SAT problem, we have variables that correspond to hypotheses (such as *developed(Microsoft, JavaProgrammingLanguage)*) and variables that correspond to facts (namely ontological facts and textual facts). A solution to the weighted MAX SAT problem should assign the truth value 1 to all facts. Therefore, we assign the value 1 to all textual facts and all ontological facts a priori. This assumes that the ontology is consistent with the rules. In case of YAGO, this is given by the construction methods (see Section 3.2.2). Furthermore, we will assume that the ontology is complete on the *type* and *means* facts. In case of YAGO, this assumption is acceptable, because all entities in YAGO have *type* and *means* relations. If *type* and *means* are fixed, this allows certain simplifications, such as an a priori computation of the disambiguation prior (as explained in the previous section). This leaves us with a partial assignment, which already assigns truth values to a large number of statements.

Approximation. The weighted MAX SAT problem is NP-hard [63]¹¹. This means that it is impractical to find an optimal solution for large instances of the problem, as it is the case in our setting. Some special cases of the weighted MAX SAT problem can be solved in polynomial time [78, 110]. However, none of them applies in our setting. Hence, we resort to using an approximation algorithm. An *approximation algorithm for the weighted MAX SAT problem* is an algorithm that, given a weighted MAX SAT problem, produces an assignment

¹¹The original SAT problem is not NP-hard if there are at most two literals per clause. The weighted and unweighted MAX SAT problems, however, are NP-hard even when each clause has no more than two literals.

for its variables that is not necessarily an (optimal) solution. The quality of that assignment is assessed by the *approximation ratio*:

DEFINITION 25: [*Approximation Ratio of an Assignment*]

Given a weighted MAX SAT problem in the form of a set \mathcal{X} of variables and a set \mathcal{C} of weighted clauses, and given a solution v_o , the *approximation ratio* of another assignment v for \mathcal{X} is the ratio

$$\sum_{\substack{c \in \mathcal{C} \\ c \text{ satisfied in } v}} w(c) \quad / \quad \sum_{\substack{c \in \mathcal{C} \\ c \text{ satisfied in } v_o}} w(c)$$

An algorithm for the weighted MAX SAT problem is said to have an *approximation guarantee* of $r \in [0, 1]$, if its output has an approximation ratio greater than or equal to r for all weighted MAX SAT problems. Many algorithms have only a weak approximation guarantee, but perform better in practice.

Approximation Algorithms. The weighted MAX SAT problem is an active area of research and numerous approximate algorithms have been proposed.¹² In some special cases of the weighted MAX SAT problem, the optimal solution can be approximated with a high approximation guarantee [140, 58]. However, our case is again too general. Out of the vast array of available methods, we focus on *greedy algorithms* here. This choice has two reasons: First, these methods are extremely simple and run in linear or quadratic time (in the total size of the clauses). Second, they will allow us to incorporate a resolution-like strategy in a straightforward way.

Greedy Algorithms. We call an algorithm *greedy*, if it assigns the variables incrementally without ever undoing its decision. One of the most prominent greedy algorithms is *Johnson's Algorithm* [80]. It is particularly simple and has been shown to have an approximation guarantee of $2/3$ [36]. However, the algorithm cannot produce assignments with an approximation ratio greater than $2/3$ if the problem has the following shape [156]: For some integer k , the set of variables is $\mathcal{X} = \{x_1, \dots, x_{3k}\}$ and the set of clauses is

$$\begin{array}{l} x_{3i+1} \vee x_{3i+2} \\ x_{3i+1} \vee x_{3i+3} \\ \neg x_{3i+1} \end{array} \quad \text{for } i = 1, \dots, k$$

This, however, is exactly the shape of clauses induced by the rule for functional relations (in negation, see Section 5.2.2). This means that Johnson's Algorithm cannot solve SOFIE's weighted MAX SAT problem optimally, if instances of functional relations appear. Since already the relation *disambiguatedAs* falls into this category, Johnson's Algorithm is less well suited for our problem. Hence, we consider a different greedy algorithm here.

FMS Algorithm. We introduce the *Functional Max Sat Algorithm* here, which is aimed at clauses induced by functional relations. The algorithm uses *unit clauses*:

¹²See [14] for a survey.

DEFINITION 26: [*Unit Clause*]

Given a set of variables \mathcal{X} , a partial assignment v on \mathcal{X} and a set of clauses \mathcal{C} on \mathcal{X} , a *unit clause* is a clause $c \in \mathcal{C}$ that is not satisfied in v and that contains exactly one unassigned literal.

Strictly speaking, a unit clause is only defined with respect to an assignment. To simplify, we will occasionally not mention the assignment explicitly, when the assignment is clear from the context. Intuitively speaking, unit clauses are the clauses whose satisfaction in the current partial assignment depends only on one single variable. Our algorithm uses them as follows:

ALGORITHM 6: Functional Max Sat (FMS)**Input:** Set of variables \mathcal{X} Set of weighted clauses \mathcal{C} **Output:** Assignment v for \mathcal{X}

```

1   $v :=$  the empty assignment
2  FOR EACH  $x \in \mathcal{X}$ 
3     $m^0(x) := \sum \{ w(c) \mid c \in \mathcal{C} \text{ unit clause, } x \in c^0 \}$ 
4     $m^1(x) := \sum \{ w(c) \mid c \in \mathcal{C} \text{ unit clause, } x \in c^1 \}$ 
5   $Q :=$  priority queue of all  $x \in \mathcal{X}$ ,
    ordered by descending  $|m^1(x) - m^0(x)|$ 
6  WHILE  $Q$  is not empty
7     $x^* := Q.dequeue()$ 
8     $v(x^*) = \lceil m^1(x^*) > m^0(x^*) \rceil$ 
9    FOR EACH unassigned  $x$  s.t.  $\exists C \in \mathcal{C} : x^* \in C, x \in C$ 
10     recompute  $m^0(x), m^1(x)$ , update priorities in  $Q$ 

```

The algorithm takes as input a set of variables \mathcal{X} and a set of weighted clauses \mathcal{C} . In order to assign a truth value to a variable x , the algorithm considers only the unit clauses in which x appears. It computes for each variable x the sum of the weights of the unit clauses in which x appears with negative polarity ($m^0(x)$, line 3). Analogously, it computes the weights of the unit clauses in which x appears positive ($m^1(x)$, line 4). In case there are no unit clauses with x , both $m^0(x)$ and $m^1(x)$ are zero. In the loop (lines 6-10), the algorithm always picks the variable x that exhibits the largest difference of $m^0(x)$ and $m^1(x)$, breaking ties arbitrarily (line 7). In case there are no unit clauses at all, $m^0(x) = m^1(x) = 0$ for all $x \in \mathcal{X}$. In this case, an arbitrary variable x is dequeued from Q . If $m^1(x) > m^0(x)$, x is assigned the truth value 1. Else x is assigned the truth value 0 (line 8). After the assignment, the unit clauses in which x appeared are no longer unit clauses. However, new unit clauses might have sprung up. Hence, all variables affected by the assignment of x (line 9) have their values $m^0(x)$ and $m^1(x)$ recomputed. This changes their priority in the priority queue Q , so Q has to be updated (line 10). This procedure is iterated until all variables are assigned (lines 6-10). Assuming that all operations on the priority queue run in logarithmic time, the algorithm runs in time $O(n \cdot m \cdot k \cdot \log(n))$, where n is the total number of variables in the clauses, k is the maximum number of variables per clause and m is the maximum number of appearances of a variable. We prove in the Appendix B.4

that the FMS Algorithm has an approximation guarantee of $1/2$:

THEOREM 4: [*Approximation Guarantee of the FMS Algorithm*]

Independent of the order in which the variables are assigned, the FMS Algorithm has an approximation guarantee of $1/2$.

One might be tempted to construct a similar greedy algorithm that simply assigns the truth value t to a variable x , if the weight of unsatisfied clauses where x appears with polarity t exceeds the weight of unsatisfied clauses where x appears with polarity $\neg t$. We call this algorithm the *Simple Algorithm* and discuss it in Appendix B.5. In summary, it would have difficulties with the clauses in the SOFIE setting.

DUC Propagation. Once the FMS Algorithm has assigned a single variable, the truth value of others might be implied by necessity. These variables are called *safe*:

DEFINITION 27: [*Safe Variable*]

Given a set of variables \mathcal{X} , a partial assignment v on \mathcal{X} and a weighted set of clauses \mathcal{C} on \mathcal{X} , an unassigned variable $x \in \mathcal{X}$ is called *safe*, if there exists a truth value $t \in \{0, 1\}$ such that the weight of all unit clauses where x appears with polarity p is larger than the weight of all unsatisfied clauses in which x appears with polarity $\neg p$:

$$\sum_{\substack{c \text{ unit clause in } v \\ x \in c^p}} w(c) \geq \sum_{\substack{c \text{ unsatisfied clause in } v \\ x \in c^{\neg p}}} w(c)$$

p is called the *safe truth value* of x .

The following theorem [101, 152] says that safe variables can be assigned their safe truth value without changing the weight of the best solution that can still be obtained:

THEOREM 5: [*Safe Truth Values are Safe*]

Let \mathcal{X} be a set of variables, let v be a partial assignment on \mathcal{X} and let \mathcal{C} be a weighted set of clauses. Let x be a safe variable with safe truth value t . Let $v_o \supseteq v$ be an assignment that extends v and maximizes the sum of the weights of the satisfied clauses. Let v'_o be a variant of v_o that assigns t to x :

$$v'_o = v_o \setminus \{x \rightarrow 1, x \rightarrow 0\} \cup \{x \rightarrow t\}$$

Then

$$\sum_{c \in \mathcal{C} \text{ satisfied in } v'_o} w(c) \geq \sum_{c \in \mathcal{C} \text{ satisfied in } v_o} w(c)$$

We provide a proof in Appendix B.6. Theorem 5 gives rise to the technique of *Dominating Unit Clause Propagation*:

ALGORITHM 7: DUC Propagation

Input: Set of variables \mathcal{X}
 Set of weighted clauses \mathcal{C}
 Partial assignment v for \mathcal{X}

Output: Modified v

- 1 **WHILE** there exists a safe variable $x \in \mathcal{X}$
- 2 $v(x) :=$ safe truth value for x

Assigning one safe variable can create new unit clauses, which can give rise to new safe variables. Theorem 5 ensures that the safe variables can be assigned in any order without worsening the best solution that can still be achieved.¹³ In particular, the theorem ensures that if a partial assignment is part of an optimal solution, the enlarged assignment produced by the DUC Propagation will still be part of an optimal solution. Unlike the FMS Algorithm, however, DUC propagation does not necessarily produce a total assignment. Some variables may be left unassigned.

FMS Algorithm and DUC Propagation. We combine the FMS Algorithm with DUC propagation by calling the DUC propagation after each assignment (Algorithm 8).

ALGORITHM 8: FMS*

Input: Set of variables \mathcal{X}
 Set of weighted clauses \mathcal{C}

Output: Assignment v for \mathcal{X}

- 1 Run Functional MAX SAT (Alg. 6) with \mathcal{X}, \mathcal{C}
- 2 After each assignment (line 8 in Functional MAX SAT):
- 3 Run DUC Propagation (Alg. 7) with $\mathcal{X}, \mathcal{C}, v$
- 4 Remove assigned variables from Q

We prove in Appendix B.7 that this modification does not change the approximation guarantee of the algorithm:

THEOREM 6: [*Approximation Guarantee of the FMS* Algorithm*]

The FMS* Algorithm has an approximation guarantee of $1/2$.

The approximation guarantee does not improve over the approximation guarantee of the FMS Algorithm, because the presence of safe variables cannot be assumed in general.

Generating Clauses. Our algorithm works on a database representation of YAGO (see Section 3.2.3). The hypotheses and the textual facts are stored in the database as well. Since our weighted MAX SAT problem will be huge, we refrain from generating all clauses explicitly. Rather, we devised an algorithm

¹³DUC Propagation subsumes the techniques of unit propagation and pure literal elimination employed by the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [48] for the SAT problem.

that, given a statement s , generates all clauses in which s appears on the fly. This procedure is given in Algorithm 9.

ALGORITHM 9: Generate Clauses

Input: Set of rules \mathcal{R}
 Set of statements \mathcal{S}
 Statement $s \in \mathcal{S}$
 Partial assignment v on \mathcal{S}
Output: Set of weighted clauses $\mathcal{C}(s)$
 Modified set \mathcal{S}

```

1   $\mathcal{C}(s) := \emptyset$ 
2  FOR EACH rule  $r \in \mathcal{R}$ 
3    FOR EACH literal  $l \in r$ 
4      IF  $s$  is not an instance of  $l$  THEN CONTINUE FOR
5       $\sigma :=$  substitution s.t.  $\sigma(l) = s$ 
6       $toDo :=$  list of literals in  $\mathcal{R}$  without  $l$ 
7      IF  $l$  positive THEN addClauses( $toDo, \sigma, (s), \mathcal{C}(s)$ )
8      ELSE addClauses( $toDo, \sigma, (\neg s), \mathcal{C}(s)$ )
9   $\mathcal{C}(s) := \mathcal{C}(s) \cup \{(\neg s)\}$ 
10 Define  $w(c)$  for all  $c \in \mathcal{C}(s)$ 

  METHOD addClauses( $toDo, \sigma, c$ , modifiable  $\mathcal{C}$ )
1  IF  $toDo = \emptyset$  THEN
2     $\mathcal{C} := \mathcal{C} \cup \{c\}$ 
3    RETURN
4   $l' := toDo.first()$ 
5  IF  $\sigma(l')$  is a statement  $\wedge \sigma(l') \notin \mathcal{S}$  THEN  $\mathcal{S} := \mathcal{S} \cup \{\sigma(l')[?]\}$ 
6  FOR EACH  $s' \in \mathcal{S}, s'$  instance of  $\sigma(l')$ 
7    IF  $v(s') = 1 \wedge l'$  positive  $\vee$ 
        $v(s') = 0 \wedge l'$  negative THEN CONTINUE FOR
8    IF  $v(s')$  undefined THEN
9      IF  $l'$  positive THEN  $c' := c \vee s'$ 
10     ELSE  $c' := c \vee \neg s'$ 
11    ELSE  $c' := c$ 
12     $\sigma' :=$  substitution s.t.  $\sigma'(l') = s'$ 
13    addClauses( $toDo \setminus l', \sigma \cup \sigma', c', \mathcal{C}$ )

```

The algorithm takes as input a set of rules \mathcal{R} (see Section 5.2.2), a modifiable set of statements \mathcal{S} , a statement s and a partial assignment v . The algorithm will return all clauses constructed from \mathcal{R} that contain s . Furthermore, the algorithm will generate all statements that appear in these clauses, but are not yet in \mathcal{S} . First, the algorithm checks all literals in all rules (lines 2-3). If s is a ground instance of a literal l in a rule (line 4), the algorithm constructs the substitution for the placeholders in l that is necessary to match s (line 5). Next, the algorithm gathers the other literals in the rule (line 6). The algorithm sorts the literals, so that literals with *disambPrior* and *patternOcc* come first. If the literal l appeared positive in the rule (i.e., un-negated to the right-hand-

side of \Rightarrow), the method `addClauses` is called with an initial clause of (s) (else with $(\neg s)$, lines 7-8). The method `addClauses` will find all possible instances of all literals in *todo*, construct the appropriate clauses and add them to \mathcal{C} . For this purpose, the method traverses the *todo* list of literals, calling itself recursively on each literal. With each recursive call, the substitution σ grows, the tentative clause c grows and the list *todo* shrinks. The method picks the first literal l' from *todo* (line 4). In some cases, the substitution σ already binds all placeholders in the literal. In this case, $\sigma(l')$ is added to the set of statements \mathcal{S} (line 5). Next, all instances of l' are considered. If an instance satisfies the clause in the current partial assignment, the clause can be ignored (line 7). If the instance has already been assigned, it can be left out of the generated clause (line 11). If the instance is unassigned, it is added to the clause (lines 9-10). Then, the method computes the new placeholder bindings σ' (line 12) and calls itself recursively (line 13). Once a clause is complete, it is added to \mathcal{C} (line 2). After all clauses have been generated, it remains to add Ockham's rule (line 9) and to compute the weights for all clauses (line 10), as described in Section 5.2.3.

The virtual relations (see Section 2.2.8) can be evaluated on the fly while the clauses are being constructed (in line 7). This also applies to hypotheses of the form $\text{type}(x, y)$, where x is a literal and y is a class. These hypotheses are handled as follows: We associate with each literal data type (see Section 2.2.4) a regular expression that determines whether a string can be an instance of that data type. Whenever a hypothesis has the form $\text{type}(x, y)$ with y being a literal data type, we check whether x matches the regular expression of y . This allows evaluating this type of hypotheses on the fly. YAGO itself does not store facts of this form.

Final Algorithm. We will now put Algorithm 9 and Algorithm 8 together to obtain Algorithm 10, which is used to compute truth values for the hypotheses.

ALGORITHM 10: Compute True Hypotheses

Input: Set of rules \mathcal{R}

Set of facts \mathcal{F}

Output: Set of statements \mathcal{S}

Assignment v on \mathcal{S}

```

1   $\mathcal{S} := \text{copy of } \mathcal{F}$ 
2  FOR EACH  $\text{disambPrior}(wic, e, z) \in \mathcal{F}$ 
3     $\mathcal{S} := \mathcal{S} \cup \{\text{disambiguatedAs}(wic, e)[?]\}$ 
4  FOR EACH pattern  $p$  appearing in  $\mathcal{F}$ 
5    FOR EACH relation  $r$  appearing in  $\mathcal{F}$ 
6      Generate clauses (Alg. 9) with  $\mathcal{R}, \mathcal{S}, \text{expresses}(p, r), \emptyset$ 
7  Run FMS* (Alg. 8) with  $\mathcal{S}$ 
8    using Clause Generation (Alg. 9) to generate clauses when needed
```

The algorithm takes as input a set \mathcal{F} of textual and ontological facts and a set \mathcal{R} of rules. We take the rules defined in Section 5.2.2. The algorithm produces a set

of statements \mathcal{S} and an assignment v for them. The algorithm first constructs the set \mathcal{S} of statements. These consist first just of the given facts \mathcal{F} (line 1). Then, the algorithm aims to add all hypotheses of the form $disambiguatedAs(wic, e)$. From inspecting \mathcal{R} , it can be seen that such a hypothesis can only be deduced in \mathcal{R} if there is a corresponding textual fact $disambPrior(wic, e, z)$ in \mathcal{F} . Hence, the $disambiguatedAs$ hypotheses can be generated in a simple loop (lines 2-3). Then, the algorithm sets out to generate the other hypotheses. A rule can only make a hypothesis true if the literal occurs positive on the right-hand-side of ' \Rightarrow '. All rules that have a positive literal on the right-hand-side of ' \Rightarrow ' have on the left-hand-side textual literals, literals with $disambiguatedAs$, and literals with $expresses$. Once the left-hand-side literals are grounded, the right-hand-side literal is grounded as well. Since we have in \mathcal{S} already all instances of textual facts and all hypotheses with the relation $disambiguatedAs$, it suffices to traverse all possible instances of literals with the relation $expresses$ in order to generate all possible hypotheses.¹⁴ This is what the algorithm does in lines 4-6. Algorithm 9 makes sure that all hypotheses are added to \mathcal{S} . Once \mathcal{S} is complete, it remains to call the FMS* Algorithm on them (lines 7-8).

5.3.3 SOFIE Algorithm

SOFIE Algorithm. We combine the adapted LEILA (Algorithm 5) and the computation of true hypotheses (Algorithm 10) as follows:

ALGORITHM 11: SOFIE

Input: Set of rules \mathcal{R}

Set of facts \mathcal{F}

Set documents \mathcal{D}

Output: Set of statements \mathcal{S}

Assignment v on \mathcal{S}

- 1 Run Adapted LEILA (Alg. 5) with \mathcal{D} and some fixed pattern size k producing textual facts \mathcal{F}'
- 2 Compute True Hypotheses (Alg. 10) with $\mathcal{R}, \mathcal{F} \cup \mathcal{F}'$

The algorithm takes as input a set of rules. We use the rules defined in Section 5.2.2. Furthermore, the algorithm requires a set of facts. These are the facts from the ontology that is to be extended. Last, the algorithm also requires a given set of documents. The algorithm parses the documents and produces textual facts (\mathcal{F}'). Based on the ontological and the textual facts, the algorithm engenders hypotheses and computes their truth values.

Afterwards, the true hypotheses can be accepted as new members of the ontology. This applies primarily to new ontological facts (i.e., facts with relations such as *bornOnDate*). Going beyond the ontological facts, it is also possible to include the new *expresses* facts in the ontology. Thus, the ontology would store which pattern expresses which relation. Now suppose that, later, SOFIE is run

¹⁴If there were rules that contain positive literals, but no literals with the relation *expresses*, then this part of the algorithm would have to be adjusted.

on a different corpus. Since the SOFIE algorithm assigns the truth value 1 to all facts from the ontology, the later run of SOFIE would adopt the *expresses* facts from the previous run. This way, SOFIE can already build on the known patterns when it analyzes a new corpus.

5.4 Experiments

We evaluated SOFIE in two settings. One setting is the extraction of information from semi-structured text in the form of Wikipedia. The second setting has the goal of extracting information from Web documents. Last, we also compared different algorithms for solving the weighted MAX SAT problem. All experiments with SOFIE were run with YAGO as the background ontology and with the rules from Section 5.2.2, unless otherwise noted. For all experiments, we used the weight $W = 100$ for the inviolable rules, $w = 1$ for the violable rules and $\varepsilon = 0.1$ for Ockham’s Razor (see Section 5.2.3). The experiments were run on a standard desktop machine with a 3GHz CPU and 3.5GB RAM. We used the PostgreSQL Database system.

5.4.1 Information Extraction from Semi-Structured Text

SOFIE shall extract information from both natural language text and semi-structured text. In this section, we study the performance of SOFIE on semi-structured text. We use Wikipedia as a canonical sample corpus. We ran two types of experiments. One type of experiments studies the performance of SOFIE under controlled conditions. The other experiment is a large scale experiment on random Wikipedia articles.

5.4.1.1 Controlled Experiments

Setting

Corpus. We are interested in the performance of SOFIE with different pattern types (sesquinary and binary) and with different degrees of structure in the corpus. We chose the domain of newspapers. This choice has 3 reasons. First, the newspaper articles all have infoboxes. This allows us to vary the proportion of articles that have infoboxes – simply by cutting out the infoboxes from some articles. Second, we can learn relations on this domain that YAGO does not know (such as *newspaperHasLanguage*). This allows us to control the amount of background knowledge from the ontology. Last, some of the articles have categories that can be exploited by SOFIE in addition to the infoboxes. We created a corpus of 100 random Wikipedia articles about newspapers.

Background Knowledge. We decided for the relations *hasLanguage*, *ownedBy* and *newspaperFoundedOnDate*, which are not present in YAGO. This allows us to control the amount of training samples that SOFIE can use. All three relations have corresponding entries in the infoboxes. YAGO knows all newspapers in our corpus. YAGO does not know the language of the newspapers, their foundation date or their owner, but it contains the respective entities. For each relation, we added 10 instances as seed pairs to YAGO.

Metrics. To evaluate the performance of SOFIE, the output has to be compared to some ground truth. To establish this ground truth, we manually extracted all facts with the relations *hasLanguage*, *ownedBy* and *foundedOnDate* from the articles. For each fact, we noted whether it can be found in the infobox or whether the article text has to be considered. We use the standard precision and recall metrics. For a given relation r , let O_r be the set of facts with relation r that have been extracted by SOFIE. Let G_r be the set of ground truth facts with relation r . Then, the precision measures the proportion of the output facts that appear in the ground truth:

$$prec(O_r, G_r) := \frac{|O_r \cap G_r|}{|O_r|}$$

Conversely, recall measures the proportion of the ground truth facts that have been found by the system:

$$rec(O_r, G_r) := \frac{|O_r \cap G_r|}{|G_r|}$$

Experiments

Experiment 1. We were interested in how SOFIE copes with information in the form of notes. By this, we mean table-like information about an entity, such as

John Lennon
Born in: Liverpool
Genre: Rock
...

This form of data is a very common scheme in semi-structured texts. Consider, for example, tabular information in a curriculum vitae or a table listing the properties of a digital camera. The Wikipedia infoboxes provide one instantiation of this scheme, in which each line contains information about the article entity without mentioning it again. Hence, the infoboxes form a good test case for our sesquinary patterns. We first ran SOFIE only on the note-like part of our corpus (i.e., only on the infoboxes). Table 13 shows the processing time.

Table 13: Processing time of the Newspaper Corpus (1a)

Process	Time	
Parsing	1 min	
Hypothesis generation	17 sec	(2,519 hypotheses)
FMS*	1 min	
Total	2 min	

We compared the output of SOFIE to the ground truth that we extracted manually from the infoboxes. Table 14 shows the results per relation, excluding the seed pairs.

Table 14: Results on the Newspaper Corpus (1a)

Relation	# Ground truth pairs	# Output pairs	# Correct pairs	Precision	Recall
<i>foundedOnDate</i>	87	87	87	100%	100%
<i>hasLanguage</i>	25	25	25	100%	100%
<i>ownedBy</i>	57	53	53	100%	94.44%

As expected, SOFIE quickly finds the infobox attributes that correspond to the target relation. SOFIE finds, for example, that the pattern “owner =” in the infoboxes identifies the owner of a newspaper. This gives SOFIE a trivial precision and recall of nearly 100%. The missing values are due to parsing problems or because the target entity was not known under the given name to YAGO. This shows that SOFIE’s model can deal in principle with semi-structured text in the form of tables. No corpus-specific adjustments of the rules or the algorithm are necessary. The accuracy of SOFIE is as high as that of specialized techniques for HTML or XML tables.

The problem becomes more interesting, if the articles contain both a structured part and an unstructured part. Hence, we ran SOFIE also on the complete corpus. Table 15 shows the processing times.

Table 15: Processing time for the Newspaper Corpus (1b)

Process	Time
Parsing	3 min
Hypothesis generation	2 min (14,078 hypotheses)
FMS*	19 min
Total	24 min

Again, we evaluated SOFIE manually, this time with respect to the information in the infoboxes and the articles. Table 16 shows our results.

Table 16: Results on the Newspaper Corpus (1b)

Relation	# Ground truth pairs	# Output pairs	# Correct pairs	Precision	Recall
<i>foundedOnDate</i>	89	41	37	90.24%	41.57%
<i>hasLanguage</i>	45	25	25	100%	55.55%
<i>ownedBy</i>	57	29	26	89.65%	45.61%

In comparison to the previous experiment, precision and recall are lower. This is due to two reasons. First, SOFIE gets distracted by the patterns in the non-infobox text. If a pattern in the infobox identifies the correct entity, but the entity appears with another, useless pattern in the article text, SOFIE is less likely to accept the correct entity. Second, the number of ground truth pairs is higher, covering also the ground truth found in the article text. SOFIE finds sesquinary patterns also in the article text (such as “*was launched on X*”). These patterns, however, sometimes also apply to other entities. For example, the pattern “*was launched on X*” appeared in the sentence “*An Irish version of the paper was launched on 6 February, 2006*”. SOFIE did not notice that the sentence does not talk about the article entity. This lowers the precision of SOFIE. Still, the values are quite good.

In summary, we have seen that sesquinary patterns can extract facts from documents that contain information in note form or table form. This shows that

the framework of SOFIE is not limited to binary patterns. Sesquinary patterns, however, produce worse results if the document contains both unstructured and semi-structured text.

Experiment 2. If one aims for optimal performance in the special case of Wikipedia, it appears reasonable to use binary patterns. This is because the tokenization with binary patterns can be adapted to Wikipedia (see Section 5.3.1). Hence, we ran SOFIE with binary patterns on our corpus. Table 17 shows the processing time.

Table 17: Processing time for the Newspaper Corpus (2)

Process	Time	
Parsing	3 min	
Hypothesis generation	1 min	(8,422 hypotheses)
FMS*	4 min	
Total	8 min	

Again, we evaluated the output of SOFIE manually with respect to the ground truth in the infoboxes and the article text. Table 18 shows our results per relation.

Table 18: Results on the Newspaper Corpus (2)

Relation	# Ground truth pairs	# Output pairs	# Correct pairs	Precision	Recall
<i>foundedOnDate</i>	89	87	87	100%	97.75%
<i>hasLanguage</i>	45	29	28	96.55%	62.22%
<i>ownedBy</i>	57	49	49	100%	85.96%

We observe that recall and precision increase slightly with respect to the experiments with the sesquinary patterns. This is due to the tokenization of Wikipedia, which inserts the article entity before each infobox attribute. This way, the binary patterns can make full use of the infobox attributes. At the same time, they are less likely than sesquinary patterns to fall prey to sentences about other entities. The binary patterns find all facts from the infoboxes. In addition, they find some facts from the article text, but not all (e.g., for *hasLanguage*). The binary patterns allow SOFIE to do part of the work that is done by the YAGO extractors: For given seed pairs, SOFIE identifies the corresponding infobox attributes. It finds the values of the attributes, type checks them and adds them to the ontology. As our results show, SOFIE can achieve a precision that is similar to the precision of our manually designed infobox harvesting methods (see Section 3.2.1.1).

Experiment 3. We have seen that SOFIE can work on semi-structured text in the form of infoboxes. Even more challenging than the extraction from infoboxes and article text is the extraction from the article text alone. To test the performance of SOFIE without infoboxes, we removed the infoboxes from half of the documents. Mimicking the content of YAGO, we chose our seed pairs from the portion of articles that did have infoboxes. We ran SOFIE and re-evaluated its output, again discounting seed pairs. Table 19 shows the results of our evaluation.

Table 19: Results on the Newspaper Corpus (3)

Relation	# Ground truth pairs	# Output pairs	# Correct pairs	Precision	Recall
<i>foundedOnDate</i>	89	78	77	98.71%	86.51%
<i>hasLanguage</i>	45	18	18	100.00%	40.00%
<i>ownedBy</i>	57	26	26	100%	45.76%

As can be seen from the results, the recall is much lower if the infoboxes are not present. Still, SOFIE manages to find information also in the articles without infoboxes. This is because SOFIE finds the category “*Newspapers established in...*”. This category indicates the year in which the newspaper was founded. Interestingly, this category did not occur in our seed pairs for *foundedOnDate*. Thus, SOFIE had no clue about the quality of this pattern. By help of the infoboxes, however, SOFIE could establish a large number of instances of *foundedOnDate*. Since many of these had the category “*Newspapers established in...*”, SOFIE accepted also the category pattern “*Newspapers established in X*” as a good pattern for the relation *foundedOnDate*. In other words, newly found instances of the target relation induced the acceptance of new patterns, which in turn produced new instances. This principle is very close to what has been proposed for DIPRE [22] and Snowball [2]. However, in contrast to such prior work, SOFIE achieves this effect without any special consideration, simply by its principle of including patterns and hypotheses in its reasoning model.

Discussion. We have seen that SOFIE can cope well with semi-structured text in Wikipedia. It would be desirable if SOFIE could extract more information from the article text. In the ideal case, SOFIE could extract the information solely from the article text, thus abandoning the dependence on infoboxes. Then, SOFIE would perform a task similar to the task performed by KYLIN [150]. Up to now, however, the performance of SOFIE on this task trails behind the performance of KYLIN, which has a recall of over 90%. This is because KYLIN is heavily tuned and tailored to Wikipedia, whereas SOFIE is a general-purpose information extractor.

5.4.1.2 Large-Scale Experiment

Corpus. To evaluate SOFIE’s performance on semi-structured text on a larger scale, we tested SOFIE on a random set of Wikipedia articles. We created a corpus of 1000 randomly selected Wikipedia articles. We took care that each article was at least 10 kB large (discounting images). We chose 14 relations that are frequent in YAGO. We also added a rule saying that the birth date and the death date of a person shall not have a difference of more than 100 years. Then we ran SOFIE. Table 20 shows the processing time.

Table 20: Processing time for the Wikipedia Corpus

Process	Time
Parsing	1:14 hours
Hypothesis generation	3:53 hours (920,090 hypotheses)
FMS*	1:16 hours
Total	6:13 hours

We evaluated SOFIE manually. Because of the size of the corpus, we could not establish the full set of ground truth facts. Therefore, we report only precision numbers and cannot give any recall values. SOFIE sometimes extracted birth years for people who already have precise birth dates in YAGO. These facts are correct output facts. However, to make our evaluation as conservative as possible, we discounted these pairs in the evaluation. Table 21 shows the results of our evaluation.

Table 21: Results on the Wikipedia Corpus

Relation	# Output pairs	# Correct pairs	Precision
<i>actedIn</i>	4	4	100%
<i>bornIn</i>	58	41	70.69%
<i>bornOnDate</i>	87	86	98.85%
<i>diedOnDate</i>	21	21	100%
<i>directed</i>	7	5	71.43%
<i>establishedOnDate</i>	16	14	87.50%
<i>hasDuration</i>	2	2	100%
<i>hasPopulation</i>	3	3	100%
<i>hasProductionLanguage</i>	3	3	100%
<i>locatedIn</i>	69	56	81.16%
<i>writtenInYear</i>	3	3	100%
<i>hasArea</i>	0	0	
<i>hasUtcOffset</i>	0	0	
<i>hasArea</i>	0	0	
Total	273	238	87.18%

The evaluation shows good results. However, the precision values are worse than in the small-scale experiment. This is due to the thematic diversity in our corpus. The documents comprised articles about people, cities, movies, books and programming languages. Our relations, in contrast, mostly apply only to one single type. For example, *bornOnDate* applies exclusively to people. Thus, the chances for examples and counterexamples for each single relation are lowered. Still, the precision values are very good. One major problem is the *bornIn* relation. For this relation, SOFIE found the category pattern “*People from X*”. In most cases, this category indeed identifies the birth place of people. In some cases, however, the category tells where people spent their childhood. This misleads SOFIE. Table 22 shows some patterns.

The patterns are of 3 types: There are text patterns, such as “*X was born in Y*”. These have been found in the article text. The second type of patterns are category patterns, such as “*X CAT ◊ from Y*”. These have been found because the tokenizer inserts the article entity (*X*) before the category string (starting with “*CAT*”). The symbol “◊” has been inserted to replace an uppercase word. This way, the pattern matches the string “*People from Y*” as well as “*Scientists from Y*” (see again Algorithm 5.3.1 for the pattern extraction). The third class of patterns has been derived from infoboxes. For example, the pattern “*X goldenglobeawards = ◊ Y*” refers to a line in the infobox saying that the article entity (*X*, inserted by the tokenizer) has won a prize (replaced by “◊” in the pattern) for a movie *Y*.

Table 22: Some Patterns in the Wikipedia Corpus

Pattern	Relation
<i>“X was born in Y”</i>	<i>bornIn</i>
<i>“X CAT \diamond from Y”</i>	<i>bornIn</i>
<i>“X goldenglobeawards = \diamond Y”</i>	<i>actedIn</i>
<i>“X was born on Y”</i>	<i>bornOnDate</i>
<i>“X was assassinated in Y”</i>	<i>diedOnDate</i>
<i>“X (b: Y”</i>	<i>bornOnDate</i>
<i>“X’s film Y”</i>	<i>directed</i>
<i>“X CAT \diamond established in Y”</i>	<i>establishedOnDate</i>
<i>“X, a Y”</i>	<i>hasProductionLanguage</i>
<i>“X, who received the Y”</i>	<i>hasWonPrize</i>
<i>“X is a town in Y”</i>	<i>locatedIn</i>
<i>“X, published in Y”</i>	<i>writtenInYear</i>

Discussion. SOFIE has extracted and validated a multitude of patterns in Wikipedia. The patterns stemmed from the article text, the categories and the infoboxes. Thereby, SOFIE harvested both the structured and the unstructured part of Wikipedia. Our evaluation shows good, although not perfect precision values. The thematic diversity of the corpus has made it difficult for SOFIE to find consistent patterns with examples and counterexamples. SOFIE still works better on topic-wise homogeneous corpora, such as the one we analyzed in the preceding section. A larger corpus could also alleviate the problem, giving SOFIE more examples and counterexamples for specific relations. We will analyze large scale corpus performance on unstructured documents in the following section.

5.4.2 Information Extraction from Web Documents

SOFIE aims at understanding natural language text. To test SOFIE’s performance on real-world, unstructured text, we conducted two types of experiments. In the first type of experiments, we limited the background knowledge contributed by YAGO. In the second type of experiment, we tested SOFIE on a large scale on documents downloaded from the Internet.

5.4.2.1 Controlled Experiments

Setting

Corpus. Through its unifying framework, SOFIE performs three tasks at the same time: disambiguation, pattern identification and logical reasoning. To study the performance on these tasks in detail, we ran SOFIE on a small corpus under controlled conditions. We opted for the corpus of newspaper articles, which we already used for the evaluation of LEILA in Section 4.3. This corpus targets the *headquarters* relation, which holds between an organization and the city of its headquarters. The choice for this corpus has 3 reasons. First, YAGO does not know the *headquarters* relation. This allows us to control the amount of samples that are available to learn the relation. Second, the corpus is of particular finesse, as nearly all city names in the United States are ambiguous.

For example, when the corpus contains a sentence saying that Microsoft is headquartered in Redmond, we expect SOFIE to decide correctly whether Microsoft is headquartered in Redmond, Washington, or Redmond, Utah or Redmond, Oregon. Last, the corpus allows us to compare the performance of SOFIE to a prototypical standard Information Extraction system, the Snowball system [2].

In the original paper, Snowball was run on a collection of some thousand documents. For a small portion of that corpus, the authors established the ground truth manually. For copyright reasons, we only had access to this small portion. It comprises 150 newspaper articles.

Background Knowledge. The performance of SOFIE depends not only on the corpus, but also on the amount of background knowledge that is available from the ontology. The less data the ontology provides, the worse SOFIE will perform. For example, if the ontology does not know that a certain entity is an organization, SOFIE cannot establish a *headquarters* fact for it. To exclude the effect of the ontology, we manually added all organizations mentioned in the articles to YAGO. These were 120 organizations, some of which were not known to YAGO (or Wikipedia) before. Likewise, we added the few cities to YAGO that did not already exist in the ontology. This gives us a clean starting condition for our experiment, in which all failures are attributed solely to SOFIE and not to the ontology. In the original paper, the Snowball system was given 5 seed pairs of an organization and a city. Since the original pairs did not appear in our set of documents, we took 5 other pairs of an organization and a city and added them to the ontology.

Ground Truth. SOFIE shall learn which organization is headquartered in which city. To evaluate the performance of SOFIE, the results of SOFIE have to be compared to some ground truth. The authors of [2] have already manually compiled a ground truth table. It contains pairs of organizations and cities. A pair of an organization and a city is included in the ground truth if it becomes evident from an article that the organization is based in that city. Note that the type of task covered by Snowball and SOFIE is different from the type of task covered by the original LEILA. While LEILA aimed at extracting *all* occurrences of an instance of the target relation, Snowball and SOFIE aim at extracting the information itself, no matter how often it is mentioned in the corpus. We have extended the original ground truth table to cover all 120 organizations mentioned in the articles. Unlike Snowball, SOFIE extracts *disambiguated* entities. Hence, our ground truth has to be defined on disambiguated entities. Therefore, we disambiguated each name in the ground truth manually.

Metric. In the original paper [2], Snowball is evaluated using the *Ideal Metric* (see again Section 4.3). This metric assumes that the target relation is a functional relation (as the *headquarters* relation is indeed). Given a set G of ground truth pairs and a set O of output pairs, precision and recall under the ideal metric are defined as follows:

$$recall = \frac{|G \cap O|}{|G|}$$

$$precision = \frac{|G \cap O|}{|\{ \langle e_1, e_2 \rangle \in O \mid \exists e'_2 : \langle e_1, e'_2 \rangle \in G \}|}$$

Note the non-standard definition of precision: Precision is not computed with respect to the number of total output pairs $|O|$. Rather, it is computed with respect to those output pairs that have a corresponding entry in the ground truth. We call these pairs the *relevant pairs*. For example, if the ground truth contains the pair Microsoft/Redmond, then any output pair having Microsoft as its first argument is relevant. Any pair that contains an entity not mentioned in the ground truth (say, the number 42) is irrelevant. By computing precision only with respect to the relevant output pairs, this metric favors the system.

Evaluation. The output of SOFIE can be compared directly to the ground truth. As in the original Snowball paper, we accept the US state as the headquarters of some organization, if the organization is headquartered in a city in that state. The output of Snowball on the collection was provided for us by the author, Eugene Agichtein. This allowed us to compare our results to the best possible output of Snowball with the tuning done by the author. Snowball produces one output pair for each occurrence of an organization and a headquarters. These pairs may contain the same organization multiple times. Among these pairs, the pair with the highest confidence score has to be chosen. It is non-trivial to determine whether two output pairs of Snowball concern the same organization. For example, do “*Virgin Inc.*” and “*Virgin Corp.*” refer to the same company?¹⁵ In the original paper, this record linkage was done by automated means. Since we did not have these tools available, we did the record linking manually. This yields one headquarters for each organization. The output of Snowball is not canonicalized. That means that Snowball will produce the headquarters “*Redmond*” without noting to which Redmond it refers. To give Snowball the greatest possible advantage over SOFIE, we accepted *any* name of the true headquarters entity as correct. Thus, we required no disambiguation from Snowball, while we expected it from SOFIE.

Experiments

Experiment 1. To run SOFIE with minimal background knowledge, we first ran it only with the *isHeadquartersOf* relation. This relation is the inverse of the original relation *headquarteredIn*. The *isHeadquartersOf* relation is not functional. Thus, SOFIE has no counterexamples. Table 23 shows the time SOFIE needed to process the 150 documents (500kB).

Table 23: Processing time of the Snowball Corpus (1)

Process	Time
Parsing	2 min
Hypothesis generation	22 min (719 hypotheses)
FMS*	20 sec
Total	25 min

¹⁵The task of determining whether two words refer to the same entity is different from the task of disambiguation. The task of disambiguation considers only one single word (such as “*Redmond*”) and seeks to identify the entity it refers to, out of a set of possible entities. For this purpose, the disambiguation process needs to know the meanings of a word. This is not necessary for the task described here.

Table 24 shows the results of the evaluation.

Table 24: Results on the Snowball Corpus (1)

	# Ground truth pairs	# Output pairs	# Rel. pairs	# Correct pairs	Precision (ideal)	Recall
Snowball	120	429	65	37	56.69%	30.89%
SOFIE	120	35	35	32	91.43%	24.32%

Snowball achieves a reasonable precision and recall. The numbers are in tune with our findings in Section 4.3. They vary slightly because we have enlarged the ground truth table. SOFIE achieves a lower recall than Snowball. However, SOFIE achieves a much higher precision than Snowball – even though SOFIE faced the additional task of disambiguation. In fact, the 3 cases where SOFIE fails are difficult cases of disambiguation, where “*Dublin*” does not refer to the Irish capital, but to a city in Ohio.

Experiment 2. Now, we wanted to see how semantic information influences SOFIE. For this purpose, we added the original *headquarteredIn* relation, which is the inverse relation of *isHeadquartersOf*. We added a rule stating that whenever *X* is the headquarters of *Y*, *Y* is headquartered in *X*. Furthermore, we made *headquarteredIn* a functional relation, so that one organization is only headquartered in one location. Table 25 shows the time SOFIE needed to process the corpus with the additional rules.

Table 25: Processing time of the Snowball Corpus (2)

Process	Time
Parsing	2 min (as before)
Hypothesis generation	27 min (1165 hypotheses)
FMS*	31 sec
Total	30 min

Table 26 shows the results of our evaluation.

Table 26: Results on the Snowball Corpus (2)

	# Ground truth pairs	# Output pairs	# Rel. pairs	# Correct pairs	Precision (ideal)	Recall
SOFIE	120	46	46	42	91.30%	31.08%

Adding the inverse relation has allowed SOFIE to find patterns, in which the organization precedes the headquarter (such as in “*Microsoft, a Redmond-based company*”). This has increased recall to the level of Snowball’s recall. At the same time, the functional constraint has kept SOFIE’s precision at the same high level.

In summary, SOFIE achieves a YAGO-like precision of above 90 percent. This is much higher than Snowball’s precision, even though Snowball was trained on a much larger corpus. SOFIE cannot find all pairs of an organization and its headquarters. However, in the inherent trade-off between precision and recall, precision is usually valued more than recall for the purpose of ontology construction. Given that SOFIE did not just extract pairs of names, but pairs of disambiguated entities, SOFIE’s performance is remarkable.

Discussion. In general, both systems have difficulties achieving a recall of 100%. In some cases, this is due to non-trivial wording. For example, one article says that “*Companies whose share rose include Danbury-based specialty maker UCAR International Inc.*”. In this sentence, the company (UCAR International) and the headquarters (Danbury) appear separated by the words “*specialty maker*”, which are unlikely to appear in a pattern. Thus, SOFIE does not find this instance. This problem can be addressed by linguistic processing, as is done in LEILA. Sometimes, the documents express the relationship between an organization and its headquarters only implicitly. For example, some news articles are devoted to a single company (e.g., Sun Microsystems). They mention the name of the company only once in the beginning of the article and refer to it in the rest of the article as “*the company*” or “*the software maker*”. If the headquarters is mentioned in some other place in the article, pattern matching systems often have difficulties establishing the connection between the company name and the headquarters name. This problem could be addressed in several ways. If it is clear that the document is only about one company, then sesquinary patterns or KYLIN’s strategy [150] could help. This, however, risks falling prey to sentences about other entities, as we have seen in Section 5.4.1.1. In certain cases, also a linguistic analysis can help. In particular, it might help to use anaphora resolution and noun phrase resolution (i.e., finding the entity that a common noun such as “*the company*” refers to in a certain context). LEILA does this to a limited extent. It achieves a decent recall on the Snowball corpus (see Section 4.3.2.2). In some cases, redundancy can help. If the information cannot be extracted from one document, it can possibly be extracted from another. This is the philosophy of the free scope systems discussed in Section 4.1.2. Redundancy, however, cannot always be assumed. In some cases, temporal reasoning is necessary to figure out the headquarters of a company. For example, one article states that “*Safeway Inc. will move its headquarters to Pleasanton in about four or five months.*”. SOFIE cannot yet deal with this type of information. The TOB system [159] makes advances in this direction.

5.4.2.2 Large Scale Experiment

Setting

Corpus. We wanted to see how SOFIE performs on a large set of unstructured documents. To allow SOFIE to find coherent patterns, it is reasonable to choose a thematic domain. We decided for the domain of biographies, because this domain is particularly rich in factual information. There exist thousands of biographies on the Web, but it is hard to gather them. Hence, we opted for the group of US Senators (past and present). For US Senators, there exist numerous biographies on the Web. We picked 400 US Senators at random from YAGO. We used Google to retrieve 10 biographies for each senator (less, if the pages could not be accessed or were not in HTML). We excluded pages from Wikipedia. This resulted in 3440 HTML files. Extracting information from these files is a particularly challenging endeavor, for several reasons. First, the documents are arbitrary, unstructured documents from the Web. They are structurally extremely heterogeneous, containing, for example, tables, lists, advertisements, and occasionally also error messages. Some documents are not

biographies at all; others are biographies of several people in one document. Second, the disambiguation is particularly interesting. Many people mentioned in the biographies have highly ambiguous names. For example, there was one senator called James Watson, but YAGO knows 13 people with this name. Worse, some biographies that we downloaded may not be about the intended senator, but about another person with the same name. Thus, the ontological information extraction faces a serious challenge concerning the disambiguation.

Relations. We wanted to extract the birth date and place of the senators, the state in which they worked and the date and place of death (for those who died). We added a rule saying that the birth date and the death date of a person shall not have a difference of more than 100 years.

Experiment

Runtime. As explained in Section 5.3.3, we ran SOFIE in 5 batches of 20,000 pattern occurrences, keeping the true hypotheses and the patterns from the previous iteration for the next one. Table 27 shows the runtime of SOFIE.

Table 27: Processing time of the Biography Corpus

Process	Time (total)	Avg. Time per Batch
Parsing	7 hours	1:25 hours
Hypothesis generation	6:15 hours	1:15 hours
FMS*	2:35 hours	31 min
Total	15:50 hours	

Evaluation. We evaluated the results manually by checking each fact on Wikipedia. By checking the fact in Wikipedia instead of in the original biographies, we could also see whether the entities have been disambiguated correctly. This is because YAGO uses the entity identifiers from Wikipedia. Because of the size of the corpus, we could not establish the full set of ground-truth facts. Therefore, we only report precision values. SOFIE sometimes extracted birth years for people who already have precise birth dates in YAGO. These facts are correct output facts. However, to make our evaluation as conservative as possible, we discounted these pairs in the evaluation. Likewise, we discounted output facts about birth years in the evaluation, if a more precise birth date was also found. In 3 cases, we could not establish the ground truth at all. We discounted these output facts as well. Table 28 shows the results of the evaluation.

Table 28: Results on the Biography Corpus

Relation	# Output pairs	# Correct pairs	Precision
<i>politicianOf</i>	339	≈ 322	94.99%
<i>bornOnDate</i>	191	168	87.96%
<i>bornIn</i>	119	104	87.40%
<i>diedOnDate</i>	66	65	98.48%
<i>diedIn</i>	29	4	13.79%
Total	744	673	90.45%

For *politicianOf*, we evaluated only 200 facts, extrapolating the number of correct pairs and the precision accordingly. Our evaluation shows very good results.

SOFIE did not only extract birth dates, but also birth places, death dates and the states in which the people worked as politicians. Each of these facts comes with its particular disambiguation problems. The place of birth, for example, is often ambiguous, as many cities in the United States bear the same name. Even the birth date may come with its particular difficulties if the name of the person refers to multiple people. Thus, we can be extremely satisfied with our precision values.

SOFIE could not establish the death places correctly, though. This is due to two reasons. First, information on death places is sparse in YAGO, thus giving SOFIE fewer counterexamples. Second, the death place is rarely mentioned with the full name of the person. Rather, it is often mentioned at the end of a biography, referring to the person only by a pronoun or by the family name. Currently, these references cannot be resolved by SOFIE. The other cases where SOFIE fails are often difficult disambiguation cases or parsing problems. For example, one document mentions the German writer Friedrich Heinrich Karl de la Motte. SOFIE knows this writer only under the name of “*Friedrich Heinrich Karl de la Motte Fouqué*”. Hence, SOFIE assumes erroneously that Carl Friedrich Heinrich Graf von Wylich und Lottum is meant instead, a Prussian infantry general. In the majority of failures for birth dates, SOFIE assigns a birth date to a person that is named by her or his profession. For example, SOFIE claims that the Attorney General of India was born 1780-08-29. This is not correct, because, from an ontological point of view, *Attorney General of India* is a profession and not a person. The noun phrase “*the Attorney General of India*” can refer to multiple people over time. SOFIE mistakes the professions for people because YAGO erroneously contains these professions as instances of the class *person*. SOFIE finds plausible patterns in the documents. Table 29 lists some of them.

Table 29: Some Patterns in the Biography Corpus

Pattern	Relation
“ <i>X was a \diamond from Y</i> ”	<i>politicianOf</i>
“ <i>X represented Y</i> ”	<i>politicianOf</i>
“ <i>X was born in Y</i> ”	<i>bornIn</i>
“ <i>X was born in Y</i> ”	<i>bornOnDate</i>
“ <i>X died on Y</i> ”	<i>diedOnDate</i>
“ <i>X (b: Y</i> ”	<i>bornOnDate</i>
“ <i>X, of Y</i> ”	<i>politicianOf</i>
“ <i>X was born in Y</i> ”	<i>diedIn</i>

SOFIE finds natural patterns for birth and death dates. These include grammatical sentences and also non-grammatical patterns such as “(b:”. Note that the pattern “*X was born in Y*” can express both the birth date and the birth place, depending on the type of *Y*. For *politicianOf*, SOFIE finds very general patterns (such as “*X, of Y*”). Without the type checking, these would lead to a disastrous precision. By the type checks, however, it is made sure that *X* is a person and *Y* is a US state. Under these conditions, the pattern indeed identifies the state that the person represented. There were multiple erroneous patterns for *diedIn*, such as “*X was born in Y*”. All of them expressed the birth place rather than the death place. This explains the dismal performance of SOFIE on death places. The patterns were found for *diedIn*, because some

people were born in the same place that they died in. This made SOFIE assume that the pattern “*X was born in Y*” identifies not just the birth place, but also the death place of people. One option would be to add a rule saying that a pattern can only express a single relation. This would force SOFIE to decide whether the pattern talks about the death place or about the birth place. This, however, would also prohibit using the same pattern for both the birth date and the birth place. Another option would be to run SOFIE in larger batches. Then, counterexamples can be found before the pattern is prematurely accepted. Indeed, SOFIE does contain counterexamples for the interpretation of “*X was born in Y*” as an indication of the death place. These counterexamples, however, appeared in a later batch, when the pattern was already accepted.

Discussion. As the *diedIn* relation shows, SOFIE’s approach comes with its limitations. First, the pattern matching approach has inherent limitations, as outlined in the discussion of the previous experiment. Second, the modeling of SOFIE’s rules as a MAX SAT problem brings certain constraints. For example, the current model does not allow quantifiers on the rules. Thus, it is not possible to tell SOFIE that if there exists a death place, then there should also exist a death date. As a result, SOFIE predicts death places for many people who are in fact still alive. In general, the performance of SOFIE is better if more semantic constraints exist. For example, strong type constraints, functional constraints and also plausibility constraints on the birth and death dates can help SOFIE rule out false hypotheses. This axiomatic information still has to be provided by hand. Despite these limitations, the performance of SOFIE under these challenging conditions is remarkably good. We are not aware of any better Information-Extraction system for unstructured text from arbitrary Web sources.

5.4.3 Comparison of Weighted MAX SAT Algorithms

We were interested in the performance of the FMS* Algorithm (Algorithm 8) in comparison to other Algorithms. We ran two experiments, one in our SOFIE setting and one on general MAX SAT problems.

5.4.3.1 SOFIE Setting

To see how the FMS* Algorithm performs in our SOFIE setting, we ran the algorithm on a small corpus of 250 biography files (from Section 5.4.2). The adapted LEILA found 3795 pattern occurrences. This resulted in 16,186 hypotheses. We compared the FMS* Algorithm (Algorithm 8) to Johnson’s Algorithm [80] and to the simple greedy Algorithm outlined in Appendix B.5. Table 30 shows the results.

Table 30: Weighted MAX SAT Algorithms on Biography Files

Algorithm	Time	# Unsatisfied violable clauses (of 172,165)	# Unsatisfied inviolable clauses (of 70,820)	Weight of unsatisfied clauses (% of total)
FMS*	15 min	241	0	0.0013
Johnson	7 min	2,357	0	0.0301
Simple	7 min	2,583	0	0.0365

In general, all Algorithms perform very well: Only a tiny fraction of the rules are not satisfied. All inviolable clauses are satisfied. As expected (see Section 5.2.2), the algorithms cannot satisfy all violable rules. The FMS* Algorithm manages to satisfy the largest number of rules. It violates only one tenth of the rules that the other algorithms violate.

5.4.3.2 Benchmark Setting

We were interested in the performance of the FMS* Algorithm on general MAX SAT problems. Unfortunately, there are no widely accepted benchmarks for approximation algorithms. Hence, we opted for benchmarks for optimal algorithms. The International Conference on Theory and Applications of Satisfiability Testing¹⁶ provides several benchmarks. Each benchmark consists of a suite of weighted MAX SAT problems. We took all suites where the optimal solution was available. These are three suites of problems: (1) randomly generated weighted MAX SAT problems with 2 variables per clause, (2) randomly generated weighted MAX SAT problems with 3 variables per clause and (3) designed weighted MAX SAT problems (geared for “difficult” optimum solutions) with 3 variables per clause (see Table 31).

Table 31: Weighted MAX SAT Benchmarks

Benchmark	# Problems	Average # variables per problem	Average # clauses per problem
Random 2	90	100.00	600.00
Random 3	80	70.00	650.00
Designed 3	15	138.33	3050.87

We ran again the Algorithms FMS* (Algorithm 8), Johnson [80] and the simple algorithm from Appendix B.5. As a baseline, we ran also an algorithm that sets all variables to 1 and an algorithm that sets all variables to 0. For each problem and each algorithm, we computed the approximation ratio (i.e., the weight of the clauses satisfied by the algorithm divided by the weight of the clauses satisfied in the optimal solution, see Section 5.3.2). We averaged the approximation ratios over all problems in a suite. Table 32 shows the results.

Table 32: Weighted MAX SAT Algorithms on Benchmarks

Algorithm	Averaged approximation ratios, %		
	Random 2	Random 3	Designed 3
Johnson	86.6837	91.5369	99.9682
Simple	86.6919	91.4946	99.9682
FMS*	87.3069	92.2848	99.9702
All 1	80.2444	88.8820	0.0708
All 0	80.6011	88.7761	99.9682

The results show that all algorithms find good approximate solutions, with approximation ratios on average greater than 85%. In particular, all algorithms are better than the simple baselines. The setting of benchmarks is somewhat artificial and not designed for approximate algorithms. However, the experiments

¹⁶<http://www.maxsat07.udl.es/>

give us confidence that the FMS* Algorithm can have comparable performance to Johnson’s Algorithm.

Discussion. In general, all approximation algorithms performed well – both on the benchmarks and in our SOFIE setting. Despite their different approximations guarantees, all algorithms performed comparably in practice. In the SOFIE setting, however, the FMS* algorithm outperformed the other algorithms by a large margin. We conjecture that this is due to the high number of constraints introduced by the functional relations (see again Section 5.2.2). Each *disambiguatedAs* fact introduces such constraints. As discussed in Section 5.2.3, these constraints are the weakness of Johnson’s Algorithm. Hence, FMS* delivers better results. We would like to emphasize, though, that our goal was to find an algorithm that performs well in the SOFIE setting. In other settings, the algorithm may perform worse than Johnson’s Algorithm. The approximation guarantee of $1/2$ gives a lower bound on the performance in the general case.

5.5 Conclusion

Summary. This chapter has presented the information integration system SOFIE, which combines LEILA-like pattern-driven extraction and YAGO to find new ontological facts. SOFIE unifies the domains of information extraction and ontological reasoning. By this unification, SOFIE is able to take into account rule-based ontological constraints, such as the constraint that every person is born in at most one place. Furthermore, the model allows SOFIE to reason on disambiguations and the meanings of patterns. In SOFIE’s model, established knowledge and new knowledge interact seamlessly to identify the most plausible hypotheses. We have seen that the model of SOFIE generalizes the reasoning that is already done during the construction of YAGO. We have also seen that, in a certain sense, SOFIE subsumes the learning algorithm of LEILA on positive and negative patterns. Finally, the chapter explained how the model can be translated into a weighted MAX SAT problem and solved with a high approximation ratio. Our experiments have shown that SOFIE achieves a remarkable precision, even on unstructured Web documents. To our knowledge, SOFIE is the first approach that can extract canonicalized facts about individuals with different relations from unstructured Web corpora.

Discussion. Apart from the tokenization and the choice of the pattern type, SOFIE is completely source-independent. There is no feature-engineering, no learning with cross validation, no parameter estimation and no tuning on the level of the MAX SAT computation. If one wants to go beyond the default implementation, SOFIE’s model leaves room for variation. For example, instead of generating facts about pattern occurrences, LEILA could generate facts that tell which word occurred how often in which document. With appropriate rules, this information could be leveraged to extract facts in the spirit of the co-occurrence analysis proposed by De Boer et al. [20] (see Section 5.1.2). Similarly, LEILA could be instructed to extract source-specific information (e.g., the defining noun-phrase in the first sentence in a Wikipedia article). This would allow using Wikipedia-tailored feature models – similar to the ones used by KYLIN

[150]. Other types of textual facts are conceivable, such as facts about the co-occurrences of entities, facts about the topic of the document or facts concerning the style or language of the document. SOFIE's framework is not limited to binary patterns. Furthermore, SOFIE's model does not necessarily have to be cast into a weighted MAX SAT problem. Other options are conceivable, such as probabilistic approaches, fuzzy-logic-based approaches or Markov Logic Networks [112]. SOFIE itself provides just the framework, in which information extraction and ontological reasoning are joined.

Outlook. SOFIE is the third and last component of the information extraction system described in this thesis. Thus, our system is now complete. The system can not only extract new facts, but also assess their ontological quality and add the facts of good quality to the ontology. The system can be used to make YAGO ever larger, while maintaining its high quality. The following chapter will discuss where YAGO is already used in practice.

Chapter 6

Applications

The previous chapters have introduced the information extraction system LEILA and the ontology YAGO. They have also presented the information integration system SOFIE, which uses output produced by LEILA to enrich YAGO. This chapter will give brief overviews of our projects that already make use of YAGO. The projects include the ontological search engine NAGA, the hybrid search engine ESTER, the information extraction system TOB and the Tagbooster study on social tagging systems. Furthermore, this section will show some projects by other teams that already use YAGO. In particular, this section will point out to which ontologies YAGO already contributes.

6.1 NAGA

NAGA[81, 82]¹ is an ontological search engine for YAGO. It is the brainchild of my colleague Gjergji Kasneci², with me, Georgiana Ifrim³, Maya Ramanath⁴ and my supervisor Gerhard Weikum⁵ as co-authors. This section will first motivate the need for ontological search engines. It will then go on to presenting NAGA's query language and explaining NAGA's ranking model.

6.1.1 Ontological Search and Ranking

Problem Statement. The Internet has become a prime source of information. However, all major Internet search engines are still text-based. This means that they are restricted to finding keywords in Web pages. This is fully sufficient for simple information needs, but highly inconvenient for more advanced queries. Suppose for example that we are looking for people who are both scientists and politicians. First, it is close to impossible to formulate this query in terms of keywords. Second, the answer to this question is possibly distributed across multiple pages, so that no state-of-the-art search engine will be able to find it. In

¹<http://mpii.de/~kasneci/naga>

²<http://mpii.de/~kasneci>

³<http://mpii.de/~ifrim>

⁴<http://mpii.de/~ramanath>

⁵<http://mpii.de/~weikum>

fact, posing this query to Google (by using the keywords “scientist politician”) yields mostly news articles about science and politics.

Challenge. This example highlights the need for more explicit, unifying structures for the information of the Web. Ontologies, such as YAGO, could provide one building block. But an ontology is useless for our purpose if it cannot be queried in a user-friendly way. So the problem we have to tackle is twofold:

1. Designing a query language that allows formulating ontological queries in a convenient way. This challenge has already been addressed by a number of approaches⁶, including our own YAGO approach from Section 2.2.8. One of the most prominent approaches is SPARQL [148]. However, neither SPARQL nor our simple YAGO query language would allow us to capture the transitivity in the *subClassOf*-relation. Hence, we cannot express that a scientist whom we are looking for may be related to the class *scientist* by an arbitrary number of *subClassOf* edges in the ontology.
2. Designing a ranking model on answers to ontological queries. An ontological search engine that cannot rank its answers according to importance would be highly inconvenient to use. However, the ranking of ontological graphs is a completely unexplored field.

NAGA addresses both of these challenges by defining first a powerful query language and second an ontologically sensitive ranking model. The NAGA system is fully implemented and employs YAGO as a knowledge base.

6.1.2 Query Language

Queries. NAGA allows asking queries on an ontology. NAGA assumes that the ontology is a directed labeled graph. The nodes of this graph are entities and the edges are binary relations between them. Since YAGO can be projected into such a model by omitting the fact identifiers, we use YAGO as the knowledge base for NAGA. The query language of NAGA is graph-based as well: The user formulates a query in the form of an ontological graph, in which some nodes or edges are labeled with variables. Figure 10 shows how the query about scientists and politicians can be formulated in this way:

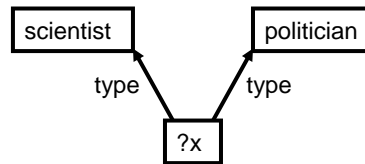


Figure 10: A Simple NAGA Query

⁶See the full NAGA paper [81] for an overview.

Answers. An answer to such a query is a subgraph of the ontology that matches the query graph. Such an answer will instantiate the variables of the query, for example as follows:

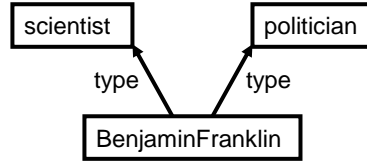


Figure 11: An Answer to the Simple NAGA Query

Advanced Queries. In this sense, NAGA’s query language is similar to SPARQL. But now suppose that, in the ontology, *BenjaminFranklin* is not connected directly to the class *scientist*. Rather, there may be intermediate classes such as *physicist* and *naturalScientist*. It is impossible to formulate the query in SPARQL so that it matches this transitive structure. The NAGA query language, however, allows labeling edges not just with simple relations (such as *type*) but also with *regular expressions* over relations⁷. In the example, the edge between *?x* and *scientist* could be labeled with the regular expression

*type subClassOf**

Then, this edge would match the path

<i>BenjaminFranklin</i>	<i>type</i>	<i>physicist</i>
	<i>subClassOf</i>	<i>naturalScientist</i>
	<i>subClassOf</i>	<i>scientist</i>

NAGA’s query language does not only allow finding paths in the ontology that match regular expression, but also finding arbitrary paths⁸. This permits, for example, to ask for the ontological connection between two people or for the similarities between two countries. YAGO’s query language (see Section 2.2.8) is a simplified version of NAGA’s language, extended by support for fact identifiers.

6.1.3 Ranking

Desiderata. A query can have multiple answers, even hundreds or thousands of answers. Thus, it is essential to rank these answers by importance. The ranking of ontological graphs is a completely unexplored area. We identified three key desiderata that should influence the ranking:

1. **Confidence:** An answer should be ranked higher if it contains facts that have high confidence values.

⁷[6] has presented a similar idea at the same conference as NAGA[82].

⁸See the full NAGA paper [81] for a formal definition of the query language.

2. **Informativeness:** An answer should be ranked higher if a variable has been instantiated by an entity that is important in the context of the query. For example, when the user asks for physicists, important physicists should be ranked first.
3. **Compactness:** Short answers should be preferred. For example, if paths of different length match a regular expression, then the most compact one should be ranked higher.

Probabilistic Model. The NAGA ranking model incorporates these three desiderata⁹. It is inspired by *language models* as used in information retrieval [69, 158]. In the NAGA ranking model, one tries to estimate the probability of a certain answer graph g , given that a certain query q was asked, $P(g|q)$. For tractability, we assume independence between the edges of g , thus obtaining

$$P(g|q) = \prod_{g_i \text{ edge in } g} P(g_i|q)$$

Confidence. We express $P(g_i|q)$ by a mixture of two distributions:

$$P(g_i|q) = \alpha P(g_i) + (1 - \alpha) \tilde{P}(g_i|q)$$

$P(g_i)$ is the query-independent probability that the edge of the answer graph is a correct fact. As discussed in Section 3.3, $P(g_i)$ is exactly the confidence value stored in YAGO for the fact g_i .

Informativeness. Now we turn to the query-dependent probability $\tilde{P}(g_i|q)$. We observe that certain answers may be more *informative* than others. For example, assume that the user asks for (famous) physicists:

?x type physicist

Then, the answer $(BenjaminFranklin, type, physicist)$ will be more satisfactory than the answer $(BobUnknown, type, physicist)$. This is because Benjamin Franklin is more important among the physicists than Bob Unknown. Speaking in terms of YAGO, the importance of a fact is reflected by the number of its witness pages. To measure the relative importance of an answer for the above query, we consider all facts of the form

?x type physicist

We consider the total number of witnesses for these facts. Next, we consider the number of witnesses for $(BenjaminFranklin, type, physicist)$. The quotient of the two will reflect the relative importance of Benjamin Franklin in the population of physicists. In general, let q_i be the edge in q that induced g_i . We write $m(q_i)$ for the set of facts in the ontology that match q_i . Then, we estimate

$$\tilde{P}(g_i|q) = \frac{\#witnesses(g_i)}{\sum_{f \in m(q_i)} \#witnesses(f)}$$

⁹In the following, we present a simplified version of the original model of [81]. The models are equivalent if each query edge corresponds to exactly one answer edge and if $\alpha = 1$ in the original model.

Ranking. By the parameter α , the first desideratum (confidence) is weighted against the second, independent, desideratum (informativeness). The third desideratum (compactness) is taken into account automatically, as the formula involves a multiplicative factor for each edge in the answer graph. A user study has shown that NAGA can provide answers that are more satisfactory than the answers delivered by standard search engines and other semantic search engines. NAGA can be tried out online at Gjergji Kasneci's homepage¹⁰.

6.2 ESTER

ESTER¹¹ [12] is hybrid search engine that combines ontological search on YAGO with full-text search on text documents. The concepts for ESTER have been devised by Holger Bast¹². ESTER is a joint project of Holger Bast, Ingmar Weber¹³ and Alexandru Chitea from the Algorithms group of the Max-Planck Institute and me from the Database group.

6.2.1 Ontological Search and Full-Text Search

The previous section 6.1 has introduced a search engine that can find facts in an ontology. Likewise, there are search engines that allow finding keywords on the Web. However, there may be queries that require a combination of the two. Suppose, for example, that the user is looking for a politician who had something to do with the pope. While the condition that the person be a politician is an ontological constraint, the condition that the person have something to do with the pope can be fulfilled best by showing that they appear together in a text document.

There are a number of projects that aim at combining ontological search with full-text search¹⁴. However, in most cases, efficiency poses a serious problem: In addition to scanning the text corpus, a hybrid search engine also has to take into account the millions of facts of the ontology. This entails that current hybrid search engines still have response times of several seconds. It would be desirable to have an engine that delivers the results at the user's fingertips. This is what ESTER does. YAGO serves as the ontological backbone of ESTER.

6.2.2 Prefix Search Engines

Word Occurrence Lists. ESTER is based on the prefix search engine from [13]. The search engine works on a *corpus*, i.e., on a collection of text documents. For a given corpus, the engine maintains a *word occurrence list*. Each entry of this list stores which word appears in which document at which position¹⁵. Consider for example the following word occurrence list:

¹⁰<http://mpii.de/~kasneci/naga>

¹¹Efficient Search on Text, Entities and Relations

¹²<http://mpii.de/~bast>

¹³<http://people.epfl.ch/ingmar.weber/>

¹⁴See the original paper [12] for a detailed overview.

¹⁵More precisely, it stores the ids of the words together with the ids of their documents in a compressed form. See [13] for details.

document	doc1.html	doc2.html	doc1.html	doc2.html	...
word	a	a	b	ba	...
position	1	17	9	42	...

This list means that the word “a” occurs in the document doc1.html at position 1 and in doc2.html at position 17. As in the example, a word occurrence list is always sorted by the lexicographic order of the words. This entails that the prefix of a word (i.e., a substring starting from the first character) defines a *range* in the list. For example, the prefix “b...” defines the range of all words that are lexicographically larger than “b” (including “b”) and lexicographically smaller than “c”. We call all words within this range (such as “barbecue”) the *completions* of that prefix.

Basic Operations. The engine supports two basic operations on word occurrence lists.

1. **Prefix Search:** For a given word occurrence list, a given set of documents and a given word range, a *prefix search* returns a sublist of the given word occurrence list that contains only the documents from the given set and only the words from the given range. For example, a prefix search allows searching for all occurrences of words starting with “Pol...” in the documents doc1.html and doc2.html. Conveniently, the result of this prefix search will also reveal all completions of “Pol...” in the given documents, such as “Politics”, “Politician” or “Polyester”.
2. **Join:** Given two word occurrence lists, a *join* computes the common words in both lists. Then, it delivers a new word occurrence list that contains the entries from the two lists that feature a common word. For example, if one word occurrence list contains occurrences of the words “Politician” and “Scientist” and a second list contains occurrences of the words “Politician” and “Musician”, then a join will first compute that “Politician” is the only word that appears in both lists. Then, it will create a new word occurrence list that contains all entries with “Politician” from the first list and all entries with “Politician” from the second list.

Bast and Weber have shown [13] how these two operations can be implemented very efficiently by using smart indexes. In fact, the search engine is so fast that results can be delivered while the user types the query¹⁶. The challenge is to transfer this technology to the hybrid search on full text and ontological facts.

6.2.3 Weaving the Ontology into the Text

Artificial Words. The idea behind ESTER is to add artificial words to the text corpus, so that full-text search on the corpus will implicitly take into account the facts from the YAGO ontology. This process takes place in two stages: First, every occurrence of an entity in the corpus is identified and replaced by the artificial word *c:e*, where *e* is the unique ontological name of the entity and

¹⁶This can be tried out with the search function at the home page of the Max-Planck Institute for Informatics, <http://mpii.de>.

c is the class that the entity is an instance of. For example, each occurrence of Tony Blair in the corpus is replaced by

```
politician:tonyBlair
```

Second, we identify for each entity in the ontology one canonical document in the corpus. If the corpus does not have a canonical document for an entity, we create a new document. To this document, we add one artificial word for each fact of the ontology that talks about the entity. For example, we add the following words to the document for Tony Blair¹⁷:

```
politician:tonyBlair
bornInYear:1953
nationality:british
```

Query Answering. ESTER can answer different types of queries: Simple keyword queries can be answered by prefix searches. For example, a prefix search for “*pope*” will deliver all documents that contain the word “*pope*”. Ontological queries can be answered as follows: Suppose we are looking for all politicians born in 1953. We first launch a prefix search for “*bornInYear:1953*”. This will deliver all documents in which “*bornInYear:1953*” appears – i.e., exactly the canonical documents of all entities born in 1953. The document about Tony Blair will be among them, as will be the document about the pianist David Benoit, for example, who was also born in 1953. On these documents, we perform another prefix search for “*politician:...*”. This search will go through the documents of entities born in 1953 and retain only those documents that contain “*politician:...*”. These documents are exactly the canonical documents about politicians born in 1953. Last, ESTER can also perform hybrid queries. For example, to ask for all politicians who had something to do with the pope, it suffices to combine a prefix search for “*pope*” with a prefix search for “*politician:...*”. In fact, ESTER delivers Tony Blair for this search, as he was granted an audience with Pope Benedict XVI in 2006.

Additional Features. By using join operations, we can also combine conditions that are satisfied in different documents. For example, we can first search for all politicians born in 1953. As we already discussed, this search involves two prefix searches and delivers the canonical documents about politicians born in 1953. Then, we can search for all documents that contain the word “*pope*” and the prefix “*politician:...*”. This search also involves two prefix searches and delivers all documents that contain the word “*pope*” and the mentioning of a politician. Joining these two lists will deliver all politicians that appear in both lists – i.e., all politicians born in 1953 that had something to do with the pope. It can be proven that the full technology of ESTER (not the simplified one outlined in this thesis) can answer arbitrary SPARQL[148] queries on the ontology.

ESTER comes with a user interface [13] that delivers immediate results while the user is still typing the query. It also proactively suggests completions. For example, when the user types “*Pope Bened*”, ESTER will suggest “*Benedict*”. Furthermore, ESTER already displays instances of classes. For example, when the user types “*politician*”, ESTER already displays a list of politicians it knows.

¹⁷This section presents a very simplified version of the actual method outlined in [12].

ESTER has response-times of less than a second. This contrasts with the performances reported in [31, 21] and also our own NAGA system [81], which are orders of magnitudes slower. ESTER can be tried out online at Holger Bast's homepage¹⁸.

6.3 TOB

TOB¹⁹ [159] is an information extraction system that can extract time information along with the facts. TOB has been driven by our visiting student Qi Zhang from the University of Science and Technology of China, with me as a co-author.

Time Information. Many real-world facts are only valid within a certain time-frame. For example, the fact that Jack Welch is the CEO of General Electric was true between 1981 and 2001 but no longer holds today. However, no current information extraction approach extracts the time dimension together with the fact²⁰. TOB is a system that can deliver this type of information.

Time Representation. TOB extends LEILA (see Chapter 4) and produces facts for YAGO (see Chapter 3). In the spirit of YAGO, time information is represented as additional facts about the primary fact (see Section 2.2.6). More precisely, the time range of a fact f is modelled by 4 relations, which each hold between the identifier of f , id_f , and a time interval (see Section 2.2.4):

id_f	<i>startsAfter</i>	t_1
id_f	<i>startsBefore</i>	t_2
id_f	<i>endsAfter</i>	t_3
id_f	<i>endsBefore</i>	t_4

This representation means that f starts after the start of t_1 , it starts before the end of t_2 , it ends after the beginning of t_3 and it ends before the end of t_4 . We call this set of facts a *fuzzy time range*. If f represents the fact that an entity is alive, we can use a fuzzy time range to describe the lifespan of the entity. This representation gives us much flexibility, because it allows us to represent partial information about the time range (such as the start interval of a fact without the end interval etc.).

Time Operations. We define two operations on fuzzy time ranges, as shown in Figure 12²¹. For each fuzzy time range, the line extends from *startsAfter* to *endsBefore*, while the box extends from *startsBefore* to *endsAfter*. Seen this way, the line represents the time during which the fact could potentially hold, while the box represents the time during which the fact holds for sure.

Assuming that A and B are two time intervals for the same fact, the *intersection operator* \cap computes a more precise time interval for the fact. It shrinks the range in which the fact could potentially hold and it widens the range it which it does hold for sure. The *inference operator* ∇ has a different purpose:

¹⁸<http://mpii.de/~bast>

¹⁹Timely Business Ontologies

²⁰See the original paper [159] for an overview of related work.

²¹See the original paper [159] for a formal definition of the operators.

Assuming that A and B are the lifespans of two entities, it computes the time range of the events in which both entities can participate.

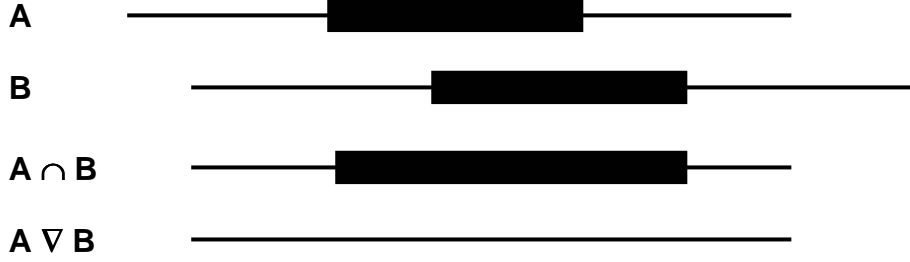


Figure 12: Operations on Fuzzy Time Ranges

TOB System. The TOB information extraction has been implemented by Qi Zhang. It extends LEILA as follows: In each linkage (see again Section 4), it identifies temporal expressions (such as dates) and temporal prepositions (such as “after” or “on”). This allows TOB to associate a fuzzy time interval with the facts. For news articles, the system also extracts the date of the publication. This allows TOB to dereference relative time expressions (such as “last week”). TOB uses fuzzy time intersection to compute a more precise time interval for a fact once more information becomes available. It uses lifespan information from the ontology to constrain the time range of an event by fuzzy time inference. An evaluation has shown that TOB achieves a precision of around 90%.

6.4 Tagbooster User Study

The goal of the Tagbooster User Study[137] was to analyze user behavior in social tagging systems. The user study was my project during my internship at Microsoft Research Cambridge under the supervision of Milan Vojnović²² and Dinan Gunawardena²³. YAGO served as a dictionary for assessing semantic properties of the tagging system.

6.4.1 Social Tagging

Tags. *Social Tagging Systems* allow users to assign keywords to content items. For example, the social tagging system del.icio.us allows users to maintain a collection of their favorite Internet links. These link collections can be shared online with other users and the users can assign *tags* (keywords) to the links. These tags are useful, for example, for organizing the link collections, finding other people’s links and categorizing links. Social tagging systems have become very popular: There exist numerous tagging systems for sharing pictures, news stories, books, blogs and videos.

²²<http://research.microsoft.com/~milanv>

²³<http://research.microsoft.com/~dinang>

The Use of Tags. From a scientific point of view, tags are interesting because they provide the key for search, classification, and possibly even information extraction for non-textual items such as songs, pictures, and videos. The usefulness of tags for these purposes, however, depends on two aspects:

1. Many tags serve purely organizational purposes (such as “*toread****”). These are of lesser use for semantic applications. It is unclear what proportion of the tags is actually “meaningful”.
2. Some social tagging systems suggest tags to the user, so that the user can simply click on the suggestion instead of typing his own tag. It is unclear in what way the user is influenced and possibly biased by these suggestions.

The TagBooster User Study sets out to shed light on these issues²⁴. It employs the YAGO ontology to define the notion of *meaningfulness* for the first item.

6.4.2 Meaningfulness

Our Dictionary. To establish whether a tag is “meaningful” or not, we compiled a dictionary as a proxy for the notion of “meaningful”. A normal dictionary of English words would not suffice, as tags can also be proper names (such as “*Madonna*”) and still be meaningful. We opted for YAGO, because by help of Wikipedia, YAGO contains the names of more than a million popular individuals. We combined YAGO with the full WordNet [59], thus obtaining a dictionary of about 2.8 million words²⁵.

Frequency Vectors. We would like to compare the tags that have been applied to a Web page to other data about the page, such as the title of the page, the category of the page in DMOZ²⁶, the keywords used to search for that page and also the content of the page itself. For this purpose, we define a framework in which all these data sources for one page are represented uniformly: For a given Web page, each data source defines a *frequency vector*. A frequency vector is a function from terms (such as tags, words or category names) to positive real numbers (such as frequency counts, weights or salience scores). For example, the content of a Web page can be represented by a frequency vector that assigns to each word the number of occurrences in the page (0 if the word does not appear in the page). Note that, because of the generality of the definition, probability distributions can also be seen as frequency vectors.

Levels of Granularity. A frequency vector can be interpreted at three levels of granularity:

1. Frequency Level
At this level, one is interested in how often a particular tag or word appears. This is the level of highest granularity.

²⁴See the original paper [137] for an overview of related work.

²⁵In our study, we considered only English words, because our target system, del.icio.us, is mostly English. However, language and culture aspects may be an interesting area of future research.

²⁶<http://dmoz.org>

2. Ranking Level

At this level, one is only interested in the relative order of terms, i.e., in establishing which term is more important than another.

3. Support Level

At this level, one is only interested in whether the frequency value of a term is non-zero. This is the coarsest granularity level of frequency vectors, which boils down to interpreting the vector as a set of terms.

Depending on the data source, certain granularity levels may be most appropriate. For example, the tag frequency vector for a Web page reveals how often a certain tag was applied to the page. Thus, the frequency level may be the most appropriate one. Our dictionary, on the other hand, can also be seen as a frequency vector: It stores whether a certain term exists in the dictionary (frequency 1) or not (frequency 0). This frequency vector calls for an interpretation on the support level.

Metrics for Frequency Vectors. The framework of frequency vectors allows seeing many standard metrics in a new light: The precision and recall metrics, for example, can be interpreted as comparing two vectors on the support level. The NDCG [76] metric essentially compares the ranking of one vector to the frequencies of another. Precision at k compares the ranking of one vector to the support of another. The cosine metric compares two vectors on the frequency level. In addition, we define another metric, the *fuzzy recall* of a frequency vector f with respect to another vector g :

$$frec(f, g) = 1 - \frac{\sum_t \max(g^*(t) - f^*(t), 0)}{\sum_t g^*(t)}$$

with

$$f^*(t) = \frac{f(t)}{\max_{t'} f(t')}; \quad g^*(t) = \frac{g(t)}{\max_{t'} g(t')}$$

The fuzzy recall and its pendant, the fuzzy precision, generalize the standard precision and recall metrics to the frequency level. These metrics allow us to compare heterogeneous sources of data on and across different levels of granularity.

Results. We applied our metrics to the public portion of the del.icio.us tagging system. We found that up to 50% of the tag applications may be “not meaningful”. This contrasts with much lower proportions of non-meaningful terms in document content and search engine queries. Furthermore, our analysis shows that the more popular a tag is, the more likely it is to be meaningful. In other words, aggregating the top tags of an item biases to filtering out the meaningful tags. This is not a priori clear as some non-meaningful words can be rather common (such as “toread”, “todo”).

Moreover, our analysis validates that the more users tagged an item, the more meaningful the most popular tags are. However, we could show that the meaningfulness increases significantly only if the item is tagged by more than 100 people. This may have consequences for small-scale tagging scenarios (such as enterprise environments). Last, our analysis indicates that tags applied to an item typically intersect more with the queries and the title than with the

content. This suggests that social tags could be a useful additional source for search applications.

6.4.3 Influence of Suggestions

Suggestions. Some social tagging systems provide *suggestions*: Whenever the user tags an item, he can either choose one of the suggested tags or type a tag himself. One purpose of the suggestions is to make tagging simpler for the user. The problem is that the suggestions may bias the user, so that the applied tags do not reflect the user's true intention. It is difficult to assess the effect of the suggestions, because a tag may happen to be the user's true intention even if it is also displayed as a suggestion.

Probabilistic Model. To assess the influence of the suggestions, we use a model introduced by Vojnović et al. [142]. It assumes that tags are applied according to the following probabilistic model: The probability that a user applies a tag t while the suggestion set S is displayed is a mixture of two distributions:

$$Pr(t|S) = \alpha_S \cdot f_S(t) + (1 - \alpha_S) \cdot g(t)$$

With probability α_S , the user decides to take a tag from the suggestion set S . $f_S(t)$ is the probability that the user chooses the tag t from the suggestion set. Consequently, $f_S(t) = 0$ for all tags $t \notin S$. With probability $1 - \alpha_S$, the user chooses an arbitrary tag, which may or may not be in S . $g(t)$ is the probability distribution over tags for this choice. We assume $f_S(t) = g(t)/g(S)$, for $t \in S$, i.e., in the cases when the sampling is from the distribution f_S , the user preference over tags is proportional to g but confined to the set S .

Estimating the Influence. Our goal is to estimate the parameter α_S , which mirrors the “persuasive power” of the suggestion set S . We consider an experiment, in which different users tag the same item. Let T_i be the set of tags applied by the i^{th} user. We define the *precision* of the experiment with respect to S as the proportion of applied tags that are in S :

$$prec = \frac{\sum_i |T_i \cap S|}{\sum_i |T_i|}$$

Now we consider two experiments: In the first (unbiased) experiment, different users apply tags to the same item without any suggestions shown. Let $prec_U$ be the precision of this experiment with respect to S . In the second (biased) experiment, different users tag the item from the first experiment while S is displayed. Let $prec_B$ be the precision of this experiment with respect to S . The desired parameter α_S can be estimated [137] as

$$\alpha_S = \frac{prec_B - prec_U}{1 - prec_U}$$

If the tags that the users applied and the suggested tags are statistically independent, then $prec_B$ and $prec_U$ will converge to the same value as the experiments go on, and we then have that α_S goes to 0, indicating no imitation. If, on the contrary, users apply tags only because they are suggested, then $prec_B$ will tend

to 1 and this will result in α_S tending to 1, indicating full imitation. α_S can be generalized to the proportion of biased tags in general, called the *imitation rate*.

Results. The goal of the internship was to conduct and evaluate a user study. More than 4000 volunteers contributed. Roughly half of them were Microsoft employees, while the others were other Internet users attracted by our advertisements for the project. The participants were asked to apply tags to Web pages under various design schemes. We are well aware that tagging in the context of a user study may differ from tagging in a social system. However, we believe that the insights that we gained from our user study give a valuable hint on the situation in real systems.

For our experiments, the imitation metric indicated that up to 30% of the tag applications have been induced purely by presence of the suggestions. Since most existing tagging systems suggest tags based on the tags that have been applied by previous users, our result suggests that the popularity of tags in existing systems may be skewed.

Furthermore, our user study revealed that users had a slight preference for the first tag in the list of suggestions. Also, users were more likely to be influenced by the suggestion set (beyond their actual preference) if the suggested tags were popular tags. However, we could also show that the bias towards the suggested tags was not due to laziness: The imitation rate stayed the same, no matter whether the suggested tags could be applied by clicking them or whether they had to be copied over by typing them. Our results are explained in detail in [137].

6.5 Other Applications

YAGO has found its way into several other projects by other teams. This section lists some of the most prominent ones.

Entity Organization. Stoyanovich et al.[130] have built 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.

Demartini [50] 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.

Information Extraction. The idea of YAGO's category heuristics has been applied by Ponzetto et al. [107] to extract ontological knowledge from Wikipedia's category system. Similarly, the KOG [151] project relies on YAGO's Wikipedia-to-WordNet mapping in order to construct a taxonomy.

Ontology Construction. YAGO is used in several major ontology projects (Figure 13). Freebase²⁷ is a community effort to gather ontological data. YAGO

²⁷<http://freebase.com>

is currently being merged into Freebase and will thus contribute to bootstrapping the project. UMBEL²⁸ is a very young project, which aims to provide a structure of subject concepts. YAGO will contribute the individuals to this structure. The Suggested Upper Model Ontology SUMO [102] is a highly axiomatized manually assembled ontology. We have merged SUMO and YAGO [49], thus combining the rich axioms of SUMO with the large number of individuals from YAGO. The Linking Open Data Project [18] aims to interconnect existing ontologies as Web services. YAGO is already available as a Web service (courtesy of Zitgist LLC.²⁹) and thus an integral part of the project. Cyc [95] is a commercial effort to create a huge semantic knowledge base. We are co-operating with the Cyc team in order to integrate data from YAGO into Cyc. DBpedia [8] is a project that aims to extract ontological data from Wikipedia. YAGO is used in DBpedia as a taxonomic backbone. It links the individuals to the WordNet hierarchy of concepts in DBpedia.

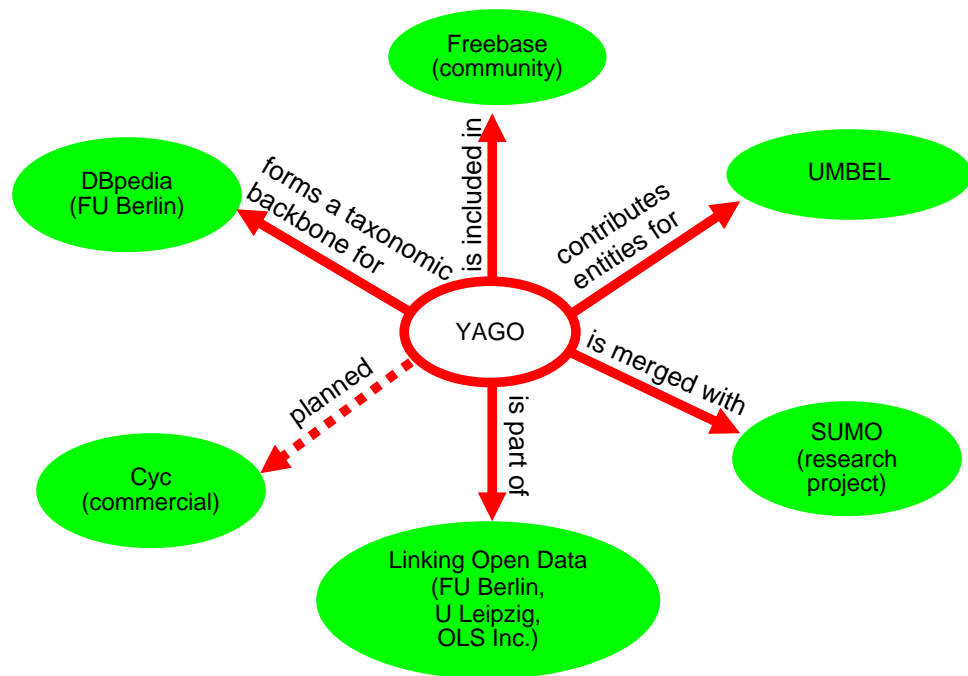


Figure 13: A Hitchhiker's Guide to Ontology

²⁸<http://www.umbel.org>

²⁹<http://www.zitgist.com>

6.6 Summary

This chapter has presented some projects that already make use of YAGO. These projects include applications as diverse as search engines, information extraction systems and analyses of social systems. In particular, this chapter has shown to which ontologies YAGO already contributes. These include logic-based projects such as SUMO as well as community-based projects such as Freebase or Semantic Web oriented projects such as DBpedia.

Chapter 7

Conclusion

Summary. This thesis has presented a new approach for the automated construction and growth of an ontology. The approach consists of three contributions: the core ontology YAGO (introduced in Chapter 3), the information extraction system LEILA (presented in Chapter 4) and the information integration framework SOFIE (in Chapter 5). Together, these systems form a powerful suite of tools, which allow not only constructing an ontology of high quality, but also maintaining and expanding it. We have seen that the YAGO ontology already enjoys a number of applications in different areas of research. In particular, YAGO is already being employed by many other major ontology projects.

Outlook. There are numerous opportunities for extending and improving this work. For example, ways could be found to discover relations automatically, so that they do not have to be provided by hand. New sources for knowledge acquisition can be tapped. New models for information extraction can be developed and analyzed. New reasoning mechanisms can be found and used. New ways of combining existing knowledge can be developed and explored. New applications for ontologies can spring up. These challenges can best be addressed in co-operation with other researchers and teams. Hence, we plan to make the code toolkit of YAGO, LEILA and SOFIE publicly available, so that new cooperations, new applications, and also new derivatives can come to life.

Discussion. Despite its size and quality, the fame of YAGO will vanish one day. Ever more comprehensive ontologies will emerge, with ever more facts and ever higher quality guarantees. Thus, it is likely that YAGO itself is some day absorbed into other projects. Today, however, the acceptance of YAGO by different communities proves something more durable: It proves the validity of our approach to ontology construction. Techniques such as type coherence checking, learning with counterexamples, and ontology-based information extraction will remain useful. More than that, the applications of YAGO have proven that ontological knowledge benefits numerous tasks in the area of computer science. And this, ultimately, might be another small step towards making computers more useful for mankind.

Appendix A

YAGO Detailed Data

A.1 Attribute Map

This appendix lists the infobox attributes together with their target relation and their markers, as explained in Section 3.2.1.1.

Attribute	Relation	Inverse	Manifold	Indirect
<i>abbreviation</i>	<i>means</i>	✓		
<i>academyawards</i>	<i>hasWonPrize</i>		✓	
<i>accessionEUdate</i>	<i>during</i>			
<i>afiawards</i>	<i>hasWonPrize</i>		✓	
<i>alias</i>	<i>means</i>	✓		
<i>alma mater</i>	<i>graduatedFrom</i>			
<i>alternatename</i>	<i>means</i>	✓		
<i>area</i>	<i>hasArea</i>			
<i>area total km</i>	<i>hasArea</i>			
<i>artist</i>	<i>created</i>	✓		
<i>author</i>	<i>wrote</i>	✓		
<i>awards</i>	<i>hasWonPrize</i>		✓	
<i>baftaawards</i>	<i>hasWonPrize</i>		✓	
<i>birth</i>	<i>bornOnDate</i>			
<i>birth date</i>	<i>bornOnDate</i>			
<i>birth name</i>	<i>means</i>	✓		
<i>birth place</i>	<i>bornIn</i>			
<i>birthdate</i>	<i>bornOnDate</i>			
<i>birthname</i>	<i>means</i>	✓		
<i>born</i>	<i>bornOnDate</i>			
<i>budget</i>	<i>hasBudget</i>			
<i>calling code</i>	<i>hasCallingCode</i>			
<i>capital</i>	<i>hasCapital</i>			
<i>cctld</i>	<i>hasTLD</i>			
<i>children</i>	<i>hasChild</i>		✓	
<i>cianame</i>	<i>means</i>	✓		
<i>citizenship</i>	<i>isCitizenOf</i>		✓	
<i>city#party</i>	<i>null</i>			

Attribute	Relation	Inv.	Man.	Ind.
combatant	participatedIn	✓	✓	
commander	isLeaderOf	✓		
commander-in-chief	isLeaderOf	✓		
common name	means	✓		
company name	means	✓		
composer	created	✓		
cover artist	madeCoverFor	✓		
creator	created	✓		
currency	hasCurrency			
date created	createdOnDate			
datebirth	bornOnDate			
death	diedOnDate			
death date	diedOnDate			
death place	diedIn			
deathdate	diedOnDate			
density km	hasPopulationDensity			
designer	created	✓		
developer	created	✓		
died	diedOnDate			
director	directed	✓		
doctoral advisor	hasAcademicAdvisor			
doctoral students	hasAcademicAdvisor	✓	✓	
duration	hasDuration			
economy#country	null			
established date	establishedOnDate			
expenses	hasExpenses			
export-goods	exports		✓	
export-partners	dealsWith		✓	
exports	hasExport			
famous works	created		✓	
father	hasChild	✓		
film#country	producedIn			
followed by	hasSuccessor			
foundation	establishedOnDate			
founded	createdOnDate			
founder	created	✓		
fullname	means	✓		
gdp	hasGDPPPP			
gdp nominal	hasNominalGDP			
gdp nominal year	during			✓
gdp ppp	hasGDPPPP			
gdp ppp year	during			✓
gdp total	hasGDPPPP			
gdp year	during			✓
genre	isOfGenre			
gini	hasGini			
gini year	during			✓
goldenglobeawards	hasWonPrize		✓	
growth	hasEconomicGrowth			
hdi	hasHDI			

Attribute	Relation	Inv.	Man.	Ind.
<i>hdi year</i>	<i>during</i>			✓
<i>headquarters</i>	<i>locatedIn</i>			
<i>height</i>	<i>hasHeight</i>			
<i>homepage</i>	<i>hasWebsite</i>			
<i>imdb id</i>	<i>hasImdb</i>			
<i>import-goods</i>	<i>imports</i>		✓	
<i>import-partners</i>	<i>dealsWith</i>		✓	
<i>imports</i>	<i>hasImport</i>			
<i>inflation</i>	<i>hasInflation</i>			
<i>influenced</i>	<i>influences</i>		✓	
<i>influenced by</i>	<i>influences</i>	✓	✓	
<i>influences</i>	<i>influences</i>	✓	✓	
<i>instrument</i>	<i>musicalRole</i>		✓	
<i>isbn</i>	<i>hasISBN</i>			
<i>known for</i>	<i>interestedIn</i>		✓	
<i>labor</i>	<i>hasLabor</i>			
<i>language</i>	<i>hasProductionLanguage</i>			
<i>language family#child</i>	<i>type</i>	✓		
<i>language family#region</i>	<i>isSpokenIn</i>		✓	
<i>leader name</i>	<i>isLeaderOf</i>	✓		
<i>lived</i>	<i>bornOnDate</i>			
<i>local name</i>	<i>means</i>	✓		
<i>location</i>	<i>locatedIn</i>			
<i>location city</i>	<i>locatedIn</i>			
<i>magnum opus</i>	<i>wrote</i>			
<i>main interests</i>	<i>interestedIn</i>		✓	
<i>mission name</i>	<i>means</i>	✓		
<i>mother</i>	<i>hasChild</i>	✓		
<i>musical artist#genre</i>	<i>null</i>			
<i>name</i>	<i>means</i>	✓		
<i>national military#amount</i>	<i>hasBudget</i>			
<i>national military#available</i>	<i>hasNumberOfPeople</i>			
<i>national military#country</i>	<i>hasMilitary</i>	✓		
<i>national motto</i>	<i>hasMotto</i>			
<i>nationality</i>	<i>isCitizenOf</i>		✓	
<i>native name</i>	<i>isNativeNameOf</i>	✓	✓	
<i>next</i>	<i>hasSuccessor</i>			
<i>next album</i>	<i>hasSuccessor</i>			
<i>nickname</i>	<i>means</i>	✓		
<i>notable ideas</i>	<i>discovered</i>		✓	
<i>null#gini</i>	<i>hasGini</i>			
<i>num employees</i>	<i>hasNumberOfPeople</i>			
<i>occupation</i>	<i>type</i>		✓	
<i>official languages</i>	<i>hasOfficialLanguage</i>		✓	
<i>official name</i>	<i>means</i>	✓		
<i>officiallang</i>	<i>hasOfficialLanguage</i>			
<i>order</i>	<i>isNumber</i>			
<i>organs</i>	<i>joined</i>			
<i>origin</i>	<i>originatesFrom</i>			
<i>pages</i>	<i>hasPages</i>			

Attribute	Relation	Inv.	Man.	Ind.
parents	hasChild	✓	✓	
party	isAffiliatedTo		✓	
percent GDP	usesGDP			
percent water	hasWaterPart			
percentwater	hasWaterPart			
place	happenedIn			
placeofbirth	bornIn			
placeofdeath	diedIn			
population	hasPopulation			
population as of	during			✓
population density km	hasPopulationDensity			
population estimate	hasPopulation			
population estimate year	during			✓
population total	hasPopulation			
populationdate	during			✓
populationyear	during			✓
poverty	hasPoverty			
preceded by	hasPredecessor			
predecessor	hasPredecessor			
premier	isLeaderOf	✓		
prev	hasPredecessor			
prizes	hasWonPrize		✓	
producer	produced	✓		
production company	produced	✓		
products	hasProduct			
reign	during			
release date	publishedOnDate			
residence	livesIn			
retired	until			✓
revenue	hasRevenue			
runtime	hasDuration			
scientist#field	interestedIn			
slogan	hasMotto			
song#length	hasDuration			
spouse	isMarriedTo		✓	
starring	actedIn	✓	✓	
successor	hasSuccessor			
term end	until			✓
term start	since			✓
time zone	inTimeZone			
totalarea km	hasArea			
turnedpro	since			✓
unemployment	hasUnemployment			
url	hasWebsite			
utc offset	hasUTCOffset			
website	hasWebsite		✓	
weight	hasWeight			
work institution	worksAt			
writer	created	✓		
writer#genre	null			

Attribute	Relation	Inv.	Man.	Ind.
<i>year end</i>	<i>until</i>			✓
<i>year start</i>	<i>since</i>			✓
<i>years active</i>	<i>during</i>			

A.2 Relations

This appendix provides a list of all relations in YAGO with their domain and range and their number of facts. This table lists all relations, including the witness relations (*foundIn*, *during* and *using*) and the trivial relations (*inUnit*, *hasValue* and *describes*). Class names that start with “yago...” have been introduced explicitly for YAGO (see Section 2.2.4). All other classes are WordNet classes.

Relation	Domain	Range	#Facts
<i>during</i>	<i>yagoFact</i>	<i>yagoTimeInterval</i>	21287651
<i>foundIn</i>	<i>yagoFact</i>	<i>yagoURL</i>	21224081
<i>using</i>	<i>yagoFact</i>	<i>program</i>	21224081
<i>means</i>	<i>yagoWord</i>	<i>entity</i>	5347523
<i>type</i>	<i>entity</i>	<i>yagoClass</i>	4505603
<i>inLanguage</i>	<i>yagoFact</i>	<i>language</i>	3563111
<i>isCalled</i>	<i>entity</i>	<i>yagoWord</i>	2185860
<i>describes</i>	<i>yagoURL</i>	<i>entity</i>	2124543
<i>familyNameOf</i>	<i>yagoWord</i>	<i>person</i>	569410
<i>givenNameOf</i>	<i>yagoWord</i>	<i>person</i>	568852
<i>bornOnDate</i>	<i>person</i>	<i>yagoDate</i>	441274
<i>subClassOf</i>	<i>yagoClass</i>	<i>yagoClass</i>	249463
<i>diedOnDate</i>	<i>person</i>	<i>yagoDate</i>	205469
<i>hasWebsite</i>	<i>entity</i>	<i>yagoURL</i>	130098
<i>hasValue</i>	<i>yagoQuantity</i>	<i>yagoNumber</i>	111961
<i>inUnit</i>	<i>yagoFact</i>	<i>unit of measurement</i>	111961
<i>establishedOnDate</i>	<i>entity</i>	<i>yagoDate</i>	110830
<i>isOfGenre</i>	<i>entity</i>	<i>yagoClass</i>	106797
<i>created</i>	<i>entity</i>	<i>entity</i>	95248
<i>hasPopulation</i>	<i>location</i>	<i>yagoNonNegativeInteger</i>	77928
<i>hasArea</i>	<i>location</i>	<i>yagoArea</i>	62720
<i>locatedIn</i>	<i>location</i>	<i>location</i>	60261
<i>hasSuccessor</i>	<i>entity</i>	<i>entity</i>	55535
<i>hasUTCOffset</i>	<i>location</i>	<i>yagoInteger</i>	52212
<i>since</i>	<i>yagoFact</i>	<i>yagoTimeInterval</i>	47714
<i>hasPopulationDensity</i>	<i>location</i>	<i>yagoDensityPerArea</i>	44628
<i>produced</i>	<i>entity</i>	<i>entity</i>	41747
<i>hasProductionLanguage</i>	<i>entity</i>	<i>language</i>	40738
<i>bornIn</i>	<i>person</i>	<i>location</i>	36189
<i>hasImdb</i>	<i>entity</i>	<i>yagoIdentifier</i>	33451
<i>hasDuration</i>	<i>entity</i>	<i>yagoDuration</i>	30791
<i>actedIn</i>	<i>person</i>	<i>entity</i>	28836
<i>until</i>	<i>yagoFact</i>	<i>yagoTimeInterval</i>	26049
<i>directed</i>	<i>person</i>	<i>entity</i>	23723
<i>hasWonPrize</i>	<i>entity</i>	<i>entity</i>	23076
<i>writtenInYear</i>	<i>entity</i>	<i>yagoDate</i>	20663
<i>hasPredecessor</i>	<i>entity</i>	<i>entity</i>	20515
<i>musicalRole</i>	<i>person</i>	<i>yagoClass</i>	15516
<i>livesIn</i>	<i>person</i>	<i>location</i>	14710
<i>diedIn</i>	<i>person</i>	<i>location</i>	13618

Relation	Domain	Range	#Facts
<i>isAffiliatedTo</i>	<i>entity</i>	<i>entity</i>	13038
<i>wrote</i>	<i>person</i>	<i>entity</i>	12469
<i>createdOnDate</i>	<i>entity</i>	<i>yagoDate</i>	12377
<i>publishedOnDate</i>	<i>entity</i>	<i>yagoDate</i>	11831
<i>originatesFrom</i>	<i>person</i>	<i>location</i>	11497
<i>influences</i>	<i>person</i>	<i>person</i>	9614
<i>hasPages</i>	<i>entity</i>	<i>yagoNonNegativeInteger</i>	9596
<i>hasMotto</i>	<i>entity</i>	<i>yagoString</i>	8309
<i>isNativeNameOf</i>	<i>yagoWord</i>	<i>location</i>	8063
<i>participatedIn</i>	<i>entity</i>	<i>entity</i>	7530
<i>politicianOf</i>	<i>person</i>	<i>location</i>	6198
<i>hasNumberOfPeople</i>	<i>entity</i>	<i>yagoNonNegativeInteger</i>	6171
<i>hasHeight</i>	<i>physical entity</i>	<i>yagoLength</i>	5926
<i>hasISBN</i>	<i>entity</i>	<i>yagoISBN</i>	5835
<i>isPartOf</i>	<i>yagoClass</i>	<i>yagoClass</i>	5022
<i>graduatedFrom</i>	<i>person</i>	<i>university</i>	4968
<i>isCitizenOf</i>	<i>person</i>	<i>country</i>	4865
<i>hasWeight</i>	<i>physical entity</i>	<i>yagoWeight</i>	4781
<i>hasChild</i>	<i>person</i>	<i>person</i>	4454
<i>isMarriedTo</i>	<i>person</i>	<i>person</i>	4208
<i>hasBudget</i>	<i>entity</i>	<i>yagoMonetaryValue</i>	4170
<i>happenedIn</i>	<i>entity</i>	<i>location</i>	3698
<i>hasRevenue</i>	<i>entity</i>	<i>yagoMonetaryValue</i>	3110
<i>isLeaderOf</i>	<i>person</i>	<i>entity</i>	2886
<i>interestedIn</i>	<i>person</i>	<i>entity</i>	2131
<i>hasAcademicAdvisor</i>	<i>person</i>	<i>person</i>	1599
<i>isNumber</i>	<i>person</i>	<i>yagoNonNegativeInteger</i>	1580
<i>worksAt</i>	<i>person</i>	<i>entity</i>	1401
<i>hasCapital</i>	<i>location</i>	<i>location</i>	1368
<i>isMemberOf</i>	<i>yagoClass</i>	<i>yagoClass</i>	1257
<i>hasProduct</i>	<i>entity</i>	<i>entity</i>	997
<i>madeCoverFor</i>	<i>person</i>	<i>entity</i>	951
<i>isSubstanceOf</i>	<i>yagoClass</i>	<i>yagoClass</i>	728
<i>hasOfficialLanguage</i>	<i>location</i>	<i>language</i>	560
<i>hasCurrency</i>	<i>location</i>	<i>entity</i>	367
<i>hasCallingCode</i>	<i>location</i>	<i>yagoCallingCode</i>	311
<i>hasGDPPPP</i>	<i>location</i>	<i>yagoMonetaryValue</i>	273
<i>hasTLD</i>	<i>location</i>	<i>yagoTLD</i>	234
<i>hasWaterPart</i>	<i>location</i>	<i>yagoProportion</i>	228
<i>hasHDI</i>	<i>location</i>	<i>yagoRationalNumber</i>	212

Relation	Domain	Range	#Facts
<i>hasGini</i>	<i>location</i>	<i>yagoRationalNumber</i>	99
<i>dealsWith</i>	<i>location</i>	<i>location</i>	98
<i>domain</i>	<i>yagoRelation</i>	<i>yagoClass</i>	94
<i>range</i>	<i>yagoRelation</i>	<i>yagoClass</i>	92
<i>discovered</i>	<i>person</i>	<i>entity</i>	87
<i>exports</i>	<i>location</i>	<i>yagoClass</i>	72
<i>hasNominalGDP</i>	<i>location</i>	<i>yagoMonetaryValue</i>	62
<i>imports</i>	<i>location</i>	<i>yagoClass</i>	53
<i>hasImport</i>	<i>location</i>	<i>yagoMonetaryValue</i>	44
<i>hasInflation</i>	<i>location</i>	<i>yagoProportion</i>	44
<i>hasEconomicGrowth</i>	<i>location</i>	<i>yagoProportion</i>	43
<i>hasExpenses</i>	<i>entity</i>	<i>yagoMonetaryValue</i>	43
<i>hasExport</i>	<i>location</i>	<i>yagoMonetaryValue</i>	41
<i>hasUnemployment</i>	<i>location</i>	<i>yagoProportion</i>	41
<i>hasLabor</i>	<i>location</i>	<i>yagoNonNegativeInteger</i>	39
<i>hasPoverty</i>	<i>location</i>	<i>yagoProportion</i>	35
<i>subPropertyOf</i>	<i>yagoRelation</i>	<i>yagoRelation</i>	6

A.3 Heuristics

This appendix lists all of our heuristics with their type and estimated accuracy. The heuristics are mostly named after their target relation.

Hueristic	Type	Accuracy
<i>hasExpenses</i>	Infobox	100.0000%
<i>hasInflation</i>	Infobox	100.0000%
<i>hasLabor</i>	Infobox	97.6744%
<i>during</i>	Infobox	97.4895%
<i>type</i>	Category	96.9434%
<i>participatedIn</i>	Infobox	96.9434%
<i>plays</i>	Infobox	96.9434%
<i>establishedOnDate 1</i>	Category	96.8429%
<i>createdOn</i>	Infobox	96.8429%
<i>originatesFrom</i>	Infobox	96.8429%
<i>worksAt</i>	Infobox	96.8429%
<i>locatedIn</i>	Category	96.7902%
<i>politicianOf</i>	Category	96.7356%
<i>actedIn</i>	Infobox	96.6208%
<i>hasArea</i>	Infobox	96.6208%
<i>hasCurrency</i>	Infobox	96.6208%
<i>diedOnDate</i>	Category	96.5603%
<i>madeCoverFor</i>	Infobox	96.5603%
<i>hasCapital</i>	Infobox	96.4975%
<i>hasFamilyName</i>	Other	96.4325%
<i>hasGivenName</i>	Other	96.4325%
<i>BornPerson</i>	Infobox	96.3870%
<i>bornInLocation</i>	Infobox	96.3650%
<i>diedOnDate</i>	Infobox	96.3650%
<i>isAffiliatedTo</i>	Infobox	96.3650%
<i>livesIn</i>	Infobox	96.3650%
<i>wrote</i>	Infobox	96.3650%
<i>bornOnDate</i>	Category	96.2949%
<i>establishedOnDate 2</i>	Category	96.2949%
<i>writtenIn Year</i>	Category	96.2949%
<i>means</i>	Infobox	96.2949%
<i>dealsWith</i>	Infobox	96.2220%
<i>directed</i>	Infobox	96.1462%
<i>establishedOnDate</i>	Infobox	96.1462%
<i>hasPopulation</i>	Infobox	96.0974%
<i>bornOnDate</i>	Infobox	96.0673%
<i>hasPoverty</i>	Infobox	96.0000%
<i>hasPredecessor</i>	Infobox	95.9902%
<i>isCitizenOf</i>	Infobox	95.9902%
<i>diedInLocation</i>	Infobox	95.9851%

Target Relation	Type	Accuracy
<i>graduatedFrom</i>	Infobox	95.8994%
<i>isLeaderOf</i>	Infobox	95.8994%
<i>influences</i>	Infobox	95.8099%
<i>interestedIn</i>	Infobox	95.7309%
<i>hasWonPrize 2</i>	Category	95.7165%
<i>hasImdb</i>	Infobox	95.7165%
<i>hasUTCOffset</i>	Infobox	95.7165%
<i>happenedIn</i>	Infobox	95.6188%
<i>hasHDI</i>	Infobox	95.6188%
<i>hasISBN</i>	Infobox	95.6188%
<i>hasPopulationDensity</i>	Infobox	95.6188%
<i>hasWaterPart</i>	Infobox	95.6188%
<i>Redirect</i>	Infobox	95.5639%
<i>sells</i>	Infobox	95.5639%
<i>hasMotto</i>	Infobox	95.5165%
<i>hasPages</i>	Infobox	95.5165%
<i>hasProductionLanguage</i>	Infobox	95.5165%
<i>Lived</i>	Infobox	95.4955%
<i>hasTLD</i>	Infobox	95.4094%
<i>hasWonPrize 1</i>	Category	95.3521%
<i>hasHeight</i>	Infobox	95.3521%
<i>produced</i>	Infobox	95.3521%
<i>hasCallingCode</i>	Infobox	95.2970%
<i>isNativeNameOf</i>	Infobox	95.2970%
<i>hasDuration</i>	Infobox	95.2770%
<i>hasWebsite</i>	Infobox	95.2770%
<i>inTimeZone</i>	Infobox	95.2093%
<i>hasBudget</i>	Infobox	95.1993%
<i>hasDoctoralAdvisor</i>	Infobox	95.1789%
<i>hasNominalGDP</i>	Infobox	95.1789%
<i>hasOfficialLanguage</i>	Infobox	95.1789%
<i>WordNetLinker</i>	Other	95.1191%
<i>hasWeight</i>	Infobox	95.1191%
<i>InfoboxType</i>	Other	95.0893%
<i>hasSuccessor</i>	Infobox	94.8615%
<i>hasChild</i>	Infobox	94.4725%
<i>isMarriedTo</i>	Infobox	94.4612%
<i>created</i>	Infobox	94.2551%
<i>hasWonPrize</i>	Infobox	94.0451%
<i>exports</i>	Infobox	93.9676%

Target Relation	Type	Accuracy
<i>hasImport</i>	Infobox	93.9676%
<i>locatedIn</i>	Infobox	93.9130%
<i>hasExport</i>	Infobox	93.7720%
<i>isOfGenre</i>	Infobox	93.6227%
<i>hasRevenue</i>	Infobox	93.5117%
<i>producedInYear</i>	Infobox	92.8529%
<i>type</i>	Infobox	91.7355%
<i>hasGDPPPP</i>	Infobox	91.2219%
<i>hasGini</i>	Infobox	91.0075%
<i>discovered</i>	Infobox	90.9829%

Appendix B

Proofs

B.1 Convergence of \rightarrow

Let \mathcal{F} be a (finite) set of fact triples, as defined in Section 2.2.5. Let \rightarrow be the rewrite system defined there (see [9] for a reference on term rewriting).

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.

Proof: All rules of the rewrite system are of the form $F \rightarrow F \cup \{f\}$, where $F \subseteq \mathcal{F}$ and $f \in \mathcal{F}$. Hence \rightarrow is monotone. Furthermore, \mathcal{F} is finite. Hence \rightarrow is finitely terminating. It is easy to see that if $F \rightarrow F \cup \{f_1\}$ and $F \rightarrow F \cup \{f_2\}$ for some $F \subseteq \mathcal{F}$ and $f_1, f_2 \in \mathcal{F}$, then

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

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

B.2 Uniqueness of the Canonical Base

THEOREM 2: [*Uniqueness of the Canonical Base*]

The canonical base of a consistent YAGO ontology is unique.

Proof: 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 $|b'| < |b|$. Here, we 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 Section 2.2.5.

LEMMA 1: [*No circular rules*]

Let y be a consistent YAGO ontology, and $\{f_1, \dots, f_n\}$ a set of facts. Then there are no sets of facts F_1, \dots, F_n , such that that $F_1, \dots, F_n \subseteq D(y)$ and

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

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

- All rules are instances of (5). In this case, $(c, \text{subClassOf}, c) \in D(y)$ for some common entity c and hence y cannot be consistent.
- All rules are instances of (1). In this case, $(c, \text{subRelationOf}, c) \in D(y)$ for some common entity c and hence y cannot be consistent.
- All rules are instances of (2). In this case, $(c, r, c) \in D(y)$ for some common entity c and relation r and $(r, \text{type}, \text{atr}) \in D(y)$ and hence y cannot be consistent.
- $n = 2$, one rule is an instance of (1), and the other an instance of (2). In this case, $(c, r, c) \in D(y)$ for some common entity c and relation r and $(r, \text{type}, \text{atr}) \in D(y)$ 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 $B = \text{range}(b)$. Let $f \in D(y)$ be a fact such that $D(y) \setminus \{f\} \rightarrow D(y)$. Then $f \notin B$.

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

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

This sequence is a sequence of rule applications, where each rule has the form $S \hookrightarrow s$, where $S \subseteq \mathcal{F}$ and $s \in \mathcal{F}$. We call S the *premise* of the rule and s its *conclusion*. We say that a fact t *contributes* to a set of facts T in the sequence B_0, \dots, B_n , if there is a sequence of rule applications r_1, \dots, 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 $D(y) \setminus \{f\} \rightarrow D(y)$, there must be a rule $G \hookrightarrow f$ with $G \subseteq D(y) \setminus \{f\}$. 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 $B \setminus \{f\}$ 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 $B = \text{range}(b)$. We define the set

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

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

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

B.3 Bound on False Labels

THEOREM 3: [*Probability of False Labeling*]

With the definitions from Section 4.2.3.2, the probability that a prototype p receives a positive label in the training phase of the adaptive kNN classifier is bounded as follows:

$$P(\#EX_p > \#CE_p) \leq 2e^{-\frac{1}{2}Na_p^2f_p^2} + 2e^{(2-a_p f_p N) \cdot (\frac{1}{2}-q_p)^2}$$

Proof: Our proof makes use of the Chernoff-Hoeffding bound: For a threshold $t \geq 0$ and independent and identically-distributed random variables X_1, \dots, X_n that follow a Bernoulli distribution with parameter $p \in [0, 1]$, the probability that the average distance of the variable values to p exceeds t is bounded by

$$P(|\sum_i \frac{X_i}{n} - p| \geq t) \leq 2 \exp(-2nt^2)$$

This bound can be generalized to any $x > pn$:

$$\begin{aligned} & P(\sum_{i=1}^n X_i \geq x) \\ = & P(\sum_{i=1}^n \frac{X_i}{n} - p \geq \frac{x}{n} - p) \\ \leq & P(|\sum_{i=1}^n \frac{X_i}{n} - p| \geq \frac{x}{n} - p) \\ \leq & 2 \exp(-2n(x/n - p)^2) \end{aligned}$$

We model the sequence of patterns as a sequence of N random events. We consider 3 Bernoulli random variables EX , CE and $CAND$, which take the value 1 if the generated pair is an example, a counterexample or a candidate, respectively. Let F_p be the Bernoulli random variable that indicates whether the pattern p occurs. We define the following random variables:

$$\begin{aligned} EX_p &= EX \cdot F_p \\ CE_p &= CE \cdot F_p \\ O &= 1 - EX_p - CE_p \end{aligned}$$

We write $\#EX_p$ and $\#CE_p$ for the total number of examples and counterexamples produced by pattern p , respectively. We write $\#O = N - \#EX_p - \#CE_p$ for the number of other events. Now we turn to proving a bound for $P(\#EX_p > \#CE_p)$. We first split this probability into two cases as follows:

$$\begin{aligned}
& P(\#EX_p > \#CE_p) \\
= & P(\#EX_p > \#CE_p | \#O \geq k) \cdot P(\#O \geq k) \\
& + P(\#EX_p > \#CE_p | \#O < k) \cdot P(\#O < k) \quad \text{for any } k \quad (*)
\end{aligned}$$

We first analyze the first summand of (*). Leaving away a factor that is between 0 and 1 yields:

$$\begin{aligned}
& P(\#EX_p > \#CE_p | \#O \geq k) \cdot P(\#O \geq k) \\
\leq & P(\#O \geq k)
\end{aligned}$$

We choose $k = \frac{1}{2} \cdot (1 + P(O)) \cdot N$. Since with this choice, $k > P(O) \cdot N$, we can apply the Chernoff-Hoeffding bound:

$$\begin{aligned}
& P(\#O \geq k) \\
\leq & 2 \exp(-2N \cdot (\frac{k}{N} - P(O))^2) \\
= & 2 \exp(-2N \cdot (\frac{1}{2} \cdot (1 + P(O)) - P(O))^2) \\
= & 2 \exp(-2N \cdot (\frac{1}{2} - \frac{1}{2}P(O))^2) \\
= & 2 \exp(-\frac{1}{2}N(1 - P(O))^2)
\end{aligned}$$

Now we turn to the second summand of (*). Introducing a case analysis yields

$$\begin{aligned}
& P(\#EX_p > \#CE_p | \#C < k) \cdot P(\#O < k) \\
= & \sum_{i=0}^{k-1} P(\#EX_p > \frac{N-i}{2} | \#C = i) \cdot P(\#O = i)
\end{aligned}$$

Again leaving away factors between 0 and 1 yields

$$\begin{aligned}
\leq & \max_{i=0 \dots k-1} P(\#EX_p > \frac{N-i}{2} | \#O = i) \cdot \sum_{i=0}^{k-1} P(\#O = i) \\
\leq & \max_{i=0 \dots k-1} P(\#EX_p > \frac{N-i}{2} | \#O = i)
\end{aligned}$$

We observe that the moment we fix $\#O$, we obtain a Binomial distribution for EX and CE with the parameter

$$q_p = \frac{P(EX_p)}{P(EX_p) + P(CE_p)}$$

Since $P(EX_p) < P(CE_p)$, we have $q_p < \frac{1}{2}$ and hence $\frac{N-i}{2} > q_p(N-i)$. This allows us to apply again the Chernoff-Hoeffding bound:

$$\begin{aligned}
& \max_{i=0 \dots k-1} P(\#EX_p > \frac{N-i}{2} | \#O = i) \\
\leq & \max_{i=0 \dots k-1} 2 \exp(-2(N-i)(\frac{1}{2} - q_p)^2) \\
= & 2 \exp(-2(N - (k-1)) \cdot (\frac{1}{2} - q_p)^2) \\
= & 2 \exp(-2(N - k + 1) \cdot (\frac{1}{2} - q_p)^2) \\
= & 2 \exp(-2(N - \frac{1}{2}(1 + P(O))N + 1) \cdot (\frac{1}{2} - q_p)^2) \\
= & 2 \exp(-(N(1 - P(O)) + 2) \cdot (\frac{1}{2} - q_p)^2)
\end{aligned}$$

Using these two bounds in (*) and observing that $1 - P(O) = a_p f_p$ yields

$$\begin{aligned}
& P(\#EX_p > \#CE_p) \\
& \leq 2 \exp(-\tfrac{1}{2}N(a_p f_p)^2) \\
& \quad + 2 \exp(-(a_p f_p N + 2) \cdot (\tfrac{1}{2} - q_p)^2) \\
& = 2 \exp(-\tfrac{1}{2}N a_p^2 f_p^2) + 2 \exp((2 - a_p f_p N) \cdot (\tfrac{1}{2} - q_p)^2)
\end{aligned}$$

B.4 Approximation Guarantee of FMS

THEOREM 4: [*Approximation Guarantee of the FMS Algorithm*]

Independent of the order in which the variables are assigned, the FMS Algorithm has an approximation guarantee of $1/2$.

Proof: We are given a weighted MAX SAT problem in the form of a set \mathcal{X} of variables and a set \mathcal{C} of weighted clauses. The FMS algorithm constructs an assignment incrementally by assigning one variable after the other. We consider two sets, which we construct incrementally as the algorithm assigns the variables: The set $\mathcal{C}^- \subseteq \mathcal{C}$ will collect unsatisfied clauses, while the set $\mathcal{C}^+ \subseteq \mathcal{C}$ will collect satisfied clauses. In each step, the algorithm selects an unassigned variable $x \in \mathcal{X}$ and chooses the truth value t , if

$$\sum_{\substack{c \in \mathcal{C} \text{ unit clause} \\ x \in c^t}} w(c) \geq \sum_{\substack{c \in \mathcal{C} \text{ unit clause} \\ x \in c^{-t}}} w(c)$$

Before t is assigned to x , we update the sets \mathcal{C}^- and \mathcal{C}^+ as follows:

$$\begin{aligned}
\mathcal{C}^+ &:= \mathcal{C}^+ \cup \{c \mid c \in \mathcal{C} \text{ unsatisfied clause, } x \in c^t\} \\
\mathcal{C}^- &:= \mathcal{C}^- \cup \{c \mid c \in \mathcal{C} \text{ unit clause, } x \in c^{-t}\}
\end{aligned}$$

Every clause that is added to \mathcal{C}^+ will be satisfied by the current assignment of t to x . Every clause in \mathcal{C}^- will be unsatisfied and cannot be satisfied by future assignments. We observe that the clauses added to \mathcal{C}^+ have a higher total weight than the clauses added to \mathcal{C}^- . Hence, the update maintains the following invariance condition:

$$\sum_{c \in \mathcal{C}^+} w(c) \geq \sum_{c \in \mathcal{C}^-} w(c)$$

Each clause $c \in \mathcal{C}$ will be added to either one of the two sets during the course of the algorithm. Hence, when the algorithm terminates, \mathcal{C}^+ contains exactly the satisfied clauses. Thus, the total weight of satisfied clauses achieved by the algorithm is:

$$\sum_{c \in \mathcal{C}^+} w(c)$$

Now consider an optimal solution v_o . Its weight is at most the weight of all clauses:

$$\sum_{c \in \mathcal{C}} w(c)$$

Hence the approximation ratio of the FMS Algorithm is

$$\begin{aligned}
& \frac{\sum_{c \in \mathcal{C}^+} w(c)}{\sum_{c \in \mathcal{C}_{v_o}} w(c)} \\
& \geq \frac{\sum_{c \in \mathcal{C}^+} w(c)}{\sum_{c \in \mathcal{C}} w(c)} \\
& \geq \frac{\sum_{c \in \mathcal{C}^+} w(c)}{\sum_{c \in \mathcal{C}^-} w(c) + \sum_{c \in \mathcal{C}^+} w(c)} \\
& \geq \frac{\sum_{c \in \mathcal{C}^+} w(c)}{\sum_{c \in \mathcal{C}^+} w(c) + \sum_{c \in \mathcal{C}^+} w(c)} \\
& \geq \frac{1}{2}
\end{aligned}$$

B.5 Simple MAX SAT Algorithm

For completeness, we also examine a simple baseline Algorithm for the weighted MAX SAT problem here.

ALGORITHM 12: Simple Algorithm

Input: Set of variables \mathcal{X}
Set of weighted clauses \mathcal{C}

Output: Assignment v for \mathcal{X}

```

1   $v :=$  the empty assignment
2  FOR EACH  $x \in \mathcal{X}$ 
3     $m^0 := \sum \{ w(c) \mid c \in \mathcal{C} \text{ unsatisfied, } x \in c^0 \}$ 
4     $m^1 := \sum \{ w(c) \mid c \in \mathcal{C} \text{ unsatisfied, } x \in c^1 \}$ 
5     $v(x) = [m^1 > m^0]$ 

```

This algorithm simply assigns the truth value t to a variable x , if the weight of unsatisfied clauses where x appears with polarity t exceeds the weight of unsatisfied clauses where x appears with polarity $\neg t$. By a similar argument as given in Appendix B.4, the Simple Algorithm also has an approximation guarantee of $1/2$. However, it can miss the optimal solution for the following type of weighted MAX SAT problem: The set of variables is $\mathcal{X} = \{X, Y, Z\}$ and the clauses are

$$\begin{array}{ll}
\neg X \vee \neg Y \vee Z & w_1 = W \\
X & w_2 = W - \varepsilon \\
Y \vee \neg Z & w_3 = W
\end{array}$$

This constellation of clauses is quite frequent in the SOFIE setting, because most rules induce clauses of the shape $\neg X \vee \neg Y \vee Z$ (see Section 5.2.3). An optimal solution could set X to 1, Y to 0, and Z to 0, gaining $3W - \varepsilon$. The proposed algorithm, however, could possibly set X to 0, earning only $2W$. One could think of ordering the variables as it is done in the FMS Algorithm, privileging variables that exhibit a large difference of clause weights. However, in the example, X does exhibit the largest difference of clause weights. Still, setting X to 0 misses the optimal solution. The FMS Algorithm, in contrast, finds the optimal solution.

B.6 Safe Variables

THEOREM 5: [*Safe Truth Values are Safe*]

Let \mathcal{X} be a set of variables, let v be a partial assignment on \mathcal{X} and let \mathcal{C} be a weighted set of clauses. Let x be a safe variable with safe truth value t . Let $v_o \supseteq v$ be an assignment that extends v and maximizes the sum of the weights of the satisfied clauses. Let v'_o be a variant of v_o that assigns t to x :

$$v'_o = v_o \setminus \{x \rightarrow 1, x \rightarrow 0\} \cup \{x \rightarrow t\}$$

Then

$$\sum_{c \in \mathcal{C} \text{ satisfied in } v'_o} w(c) \geq \sum_{c \in \mathcal{C} \text{ satisfied in } v_o} w(c)$$

Proof: Theorem 5 has first been proven for *unweighted* MAX SAT in [101] as the *Dominating Unit Clause Rule*. [152] provides a proof for weighted MAX SAT with two literals per clause. The proof can be generalized to other cases where all clause have the same number of literals. We provide a shorter, general proof here.

Let \mathcal{X} be a set of variables, let v be a partial assignment on \mathcal{X} and let \mathcal{C} be a weighted set of clauses. Let x be a safe variable with safe truth value t (see Definition 27). Let $v_o \supseteq v$ be an assignment that extends v and maximizes the sum of the weights of the satisfied clauses. Let v'_o be a variant of v_o that assigns t to x :

$$v'_o = v_o \setminus \{x \rightarrow 1, x \rightarrow 0\} \cup \{x \rightarrow t\}$$

We have to prove

$$\sum_{c \in \mathcal{C} \text{ satisfied in } v'_o} w(c) \geq \sum_{c \in \mathcal{C} \text{ satisfied in } v_o} w(c)$$

If $v_o(x) = t$, it follows $v_o = v'_o$ and the claim follows immediately. Now assume $v_o(x) = \neg t$. We first observe that the clauses satisfied by v will be satisfied by both v_o and v'_o , because $v_o \supseteq v$ and $v'_o \supseteq v$. Hence, it suffices to consider only the clauses that are not satisfied in v . We define \mathcal{C}' to be the set of clauses that are not satisfied by v . Then we have to prove

$$\sum_{c \in \mathcal{C}' \text{ satisfied in } v'_o} w(c) \geq \sum_{c \in \mathcal{C}' \text{ satisfied in } v_o} w(c)$$

Since v_o differs from v'_o only in the assignment of x , the clauses that do not contain x will either be satisfied in both v_o and v'_o or in none of them:

$$\sum_{\substack{c \in \mathcal{C}' \text{ satisfied in } v'_o \\ x \notin c}} w(c) = \sum_{\substack{c \in \mathcal{C}' \text{ satisfied in } v_o \\ x \notin c}} w(c)$$

Hence, we have to prove only

$$\sum_{\substack{c \in \mathcal{C}' \text{ satisfied in } v'_o \\ x \in c}} w(c) \geq \sum_{\substack{c \in \mathcal{C}' \text{ satisfied in } v_o \\ x \in c}} w(c) \quad (*)$$

We consider the left-hand-side of (*) and compute a lower bound for it. Given that $v'_o(x) = t$, v'_o will satisfy at least the clauses in which x appears with polarity t :

$$\sum_{\substack{c \in \mathcal{C}' \text{ satisfied in } v'_o \\ x \in c}} w(c) \geq \sum_{\substack{c \in \mathcal{C}' \\ x \in c^t}} w(c)$$

Now we consider the right-hand-side of (*). v_o will satisfy all clauses that contain x with polarity $\neg t$. It cannot satisfy the clauses that contain x with polarity t and that were unit clauses in v . However, it could potentially satisfy the non-unit clauses that contain x with polarity t . Thus, we obtain the following upper bound on the right-hand-side of (*):

$$\sum_{\substack{c \in \mathcal{C}' \\ x \in c^{\neg t}}} w(c) + \sum_{\substack{c \in \mathcal{C}' \text{ non-unit in } v \\ x \in c^t}} w(c) \geq \sum_{\substack{c \in \mathcal{C}' \text{ satisfied in } v_o \\ x \in c}} w(c)$$

Using these two bounds in (*), we have to prove

$$\sum_{\substack{c \in \mathcal{C}' \\ x \in c^t}} w(c) \geq \sum_{\substack{c \in \mathcal{C}' \\ x \in c^{\neg t}}} w(c) + \sum_{\substack{c \in \mathcal{C}' \text{ non-unit in } v \\ x \in c^t}} w(c)$$

Subtracting the second summand on both sides yields

$$\sum_{\substack{c \in \mathcal{C}' \text{ unit clause in } v \\ x \in c^t}} w(c) \geq \sum_{\substack{c \in \mathcal{C}' \\ x \in c^{\neg t}}} w(c)$$

This is the definition of x being a safe variable with safe truth value t .

Asymmetry. Note the asymmetry in the rule for safe variables: To set a variable x to a truth value t , the rule considers the *unit clauses* where x appears with polarity t and compares them to the *unsatisfied clauses* where x appears with polarity $\neg t$. One might be tempted to compare just the unsatisfied clauses where x appears with polarity t to the unsatisfied clauses where x appears with polarity $\neg t$, no matter whether the clauses are unit clauses or not. This algorithm, however, performs worse than the FMS Algorithm in certain situations that are important in the SOFIE setting, see Section B.5.

B.7 Approximation Guarantee of FMS*

THEOREM 6: [*Approximation Guarantee of the FMS* Algorithm*]

The FMS* Algorithm has an approximation guarantee of $1/2$.

Proof: We are given a weighted MAX SAT problem in the form of a set \mathcal{X} of variables and a set \mathcal{C} of weighted clauses. We have to prove that the FMS* algorithm maintains an approximation guarantee of $1/2$. As in the preceding proof in Appendix B.4 for Theorem 4, we construct two sets $\mathcal{C}^- \subseteq \mathcal{C}$ and $\mathcal{C}^+ \subseteq \mathcal{C}$, which are updated after every step of the algorithm. In each step, the algorithm will assign a truth value t to a variable x (either by DUC Propagation or by the FMS Algorithm). As in the previous proof, we update the sets \mathcal{C}^- and \mathcal{C}^+ as follows before x is assigned:

$$\begin{aligned}\mathcal{C}^- &:= \mathcal{C}^- \cup \{c \mid c \in \mathcal{C} \text{ unsatisfied clause, } x \in c^{-t}\} \\ \mathcal{C}^+ &:= \mathcal{C}^+ \cup \{c \mid c \in \mathcal{C} \text{ unit clause, } x \in c^t\}\end{aligned}$$

The variable x can be assigned in two ways:

1. x can be assigned by the FMS Algorithm. Then,

$$\sum_{\substack{c \in \mathcal{C} \text{ unit clause} \\ x \in c^t}} w(c) \geq \sum_{\substack{c \in \mathcal{C} \text{ unit clause} \\ x \in c^{-t}}} w(c)$$

2. x can be assigned by DUC Propagation. Then, by the definition of DUC Propagation,

$$\sum_{\substack{c \in \mathcal{C} \text{ unit clause} \\ x \in c^t}} w(c) \geq \sum_{\substack{c \in \mathcal{C} \text{ unsatisfied} \\ x \in c^{-t}}} w(c)$$

Thus, in both cases

$$\sum_{\substack{c \in \mathcal{C} \text{ unsatisfied clause} \\ x \in c^t}} w(c) \geq \sum_{\substack{c \in \mathcal{C} \text{ unit clause} \\ x \in c^{-t}}} w(c)$$

Hence, the following invariance condition holds throughout the course of the algorithm:

$$\sum_{c \in \mathcal{C}^+} w(c) \geq \sum_{c \in \mathcal{C}^-} w(c)$$

As shown in Appendix B.4, this entails an approximation guarantee of $1/2$.

Bibliography

- [1] Wikipedia, the Free Encyclopedia. <http://en.wikipedia.org>.
- [2] E. Agichtein, L. Gravano, J. Pavel, V. Sokolova, and A. Voskoboynik. Snowball: a prototype system for extracting relations from large text collections. *SIGMOD Records*, 30(2):612, 2001.
- [3] E. Agirre and P. Edmonds. *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology)*. Springer, Secaucus, NJ, USA, 2006.
- [4] F. Amardeilh, P. Laublet, and J.-L. Minel. Document annotation and ontology population from linguistic extractions. In *Proceedings of the international conference on Knowledge capture*, pages 161–168, New York, NY, USA, 2005. ACM.
- [5] G. Antoniou and F. van Harmelen. *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press, April 2004.
- [6] K. Anyanwu, A. Maduko, and A. Sheth. SPARQ2L: towards support for subgraph extraction queries in rdf databases. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 797–806, New York, NY, USA, 2007. ACM.
- [7] J. A. Aslam and S. E. Decatur. On the sample complexity of noise-tolerant learning. *Information Processing Letters*, 57(4):189–195, 1996.
- [8] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735, Berlin, Germany, 2007. Springer.
- [9] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, New York, NY, USA, 1998.
- [10] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In M. M. Veloso and M. M. Veloso, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2670–2676, San Francisco, USA, 2007. Morgan Kaufmann.

-
- [11] M. Banko and O. Etzioni. Strategies for lifelong knowledge extraction from the Web. In *Proceedings of the 4th international conference on Knowledge capture*, pages 95–102, New York, NY, USA, 2007. ACM.
 - [12] H. Bast, A. Chitea, F. M. Suchanek, and I. Weber. ESTER: efficient search on Text, Entities, and Relations. In C. Clarke, N. Fuhr, and N. Kando, editors, *30th International Conference on Research and Development in Information Retrieval (SIGIR'07)*, pages 671–678, Amsterdam, Netherlands, 2007. Association for Computing Machinery (ACM), Association for Computing Machinery (ACM).
 - [13] H. Bast and I. Weber. Type less, find more: Fast autocompletion search with a succinct index. In *Proceedings of the Annual International ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 364–371, New York, NY, USA, 2006. ACM.
 - [14] M. Battiti and R. Protasi. Approximate algorithms and solutions for max sat. In G. Xue, editor, *Handbook of Combinatorial Optimization*, volume 19, pages 425–430, Hingham, MA, USA, 2001. Kluwer Academic Publishers.
 - [15] A. Bechini, A. Tomasi, and J. Viotto. Enabling ontology-based document classification and management in ebxml registries. In *Proceedings of the ACM symposium on Applied computing*, pages 1145–1150, New York, NY, USA, 2008. ACM.
 - [16] S. Bergamaschi, S. Castano, S. De, C. Vimercati, S. Montanari, and M. Vincini. An intelligent approach to information integration. In *Proceedings of the Conference on Formal Ontology in Information Systems (FOIS)*, pages 253–268, Amsterdam, The Netherlands, 1998. IOS Press.
 - [17] W. Bibel, S. Hölldobler, and T. Schaub. *Wissensrepräsentation und Inferenz: eine grundlegende Einführung*. Vieweg, 1993.
 - [18] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked data on the Web. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 1265–1266, New York, NY, USA, 2008. ACM.
 - [19] S. Bloehdorn, P. Cimiano, A. Duke, P. Haase, J. Heizmann, I. Thurlow, and J. Voelker. Ontology-based question answering for digital libraries. In L. Kovcs, N. Fuhr, and C. Meghini, editors, *Proceedings of the European Conference on Research and Advanced Technologies for Digital Libraries (ECDL)*, number 4675 in Lecture Notes in Computer Science, pages 14–25, Berlin, Germany, 2007. Springer.
 - [20] V. D. Boer, M. V. Someren, and B. J. Wielinga. Extracting instances of relations from web documents using redundancy. In *Proceedings of the European Semantic Web Conference (ESWC)*, volume 4011 of *Lecture Notes in Computer Science*, Berlin, Germany, 2006. Springer.
 - [21] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. Monetdb/xquery: a fast xquery processor powered by a relational engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 479–490, New York, NY, USA, 2006. ACM.

-
- [22] S. Brin. Extracting patterns and relations from the world wide web. In *Selected papers from the International Workshop on the WWW and Databases*, pages 172–183, London, UK, 1999. Springer.
 - [23] L. D. Brown, T. T. Cai, and A. Dasgupta. Interval Estimation for a Binomial Proportion. *Statistical Science*, 16(2):101–133, May 2001.
 - [24] P. Buitelaar, D. Olejnik, and M. Sintek. A Protege Plug-In for Ontology Extraction from Text Based on Linguistic Analysis. In *Proceedings of the European Semantic Web Symposium*, pages 31–44, Berlin, Germany, 2004. Springer.
 - [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*, 2005.
 - [26] R. C. Bunescu and R. J. Mooney. A shortest path dependency kernel for relation extraction. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 724–731, Morristown, NJ, USA, 2005. Association for Computational Linguistics.
 - [27] R. C. Bunescu and M. Pasca. Using encyclopedic knowledge for named entity disambiguation. In *Proceedings of the Conference of the European Chapter of the Association of Computational Linguistics (EACL)*, pages 9–16. Association for Computer Linguistics, 2006.
 - [28] M. J. Cafarella, D. Downey, S. Soderland, and O. Etzioni. KnowItNow: fast, scalable information extraction from the web. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 563–570, Morristown, NJ, USA, 2005. Association for Computational Linguistics.
 - [29] M. J. Cafarella, C. Re, D. Suci, and O. Etzioni. Structured Querying of Web Text Data: A Technical Challenge. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 225–234. Online Proceedings, 2007.
 - [30] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the national conference on Artificial intelligence*, pages 328–334, Menlo Park, CA, USA, 1999. AAAI Press.
 - [31] P. Castells, M. Fernandez, and D. Vallet. An adaptation of the vector-space model for ontology-based information retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 19(02):261–272, 2007.
 - [32] D. Celjaska, D. Celjaska, D. M. Vargas-vera, and D. M. Vargas-vera. Ontosophie: A semi-automatic system for ontology population from text. In *Proceedings of the International Conference on Natural Language Processing*, 2004.
 - [33] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 571–580, New York, NY, USA, 2007. ACM.

-
- [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 *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 865–876, Washington, DC, USA, 2005. IEEE Computer Society.
 - [36] J. Chen, D. K. Friesen, and H. Zheng. Tight bound on Johnson’s algorithm for maximum satisfiability. *Journal of Computer System Science, Academic Press*, 58(3):622–640, 1999.
 - [37] J. Chen, D. Ji, C. L. Tan, and Z. Niu. Relation extraction using label propagation based semi-supervised learning. In *Proceedings of the International Conference on Computational Linguistics (ICCL)*, pages 129–136, Morristown, NJ, USA, 2006. Association for Computational Linguistics.
 - [38] T. Cheng, X. Yan, and K. C.-C. Chang. EntityRank: searching entities directly and holistically. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 387–398. VLDB Endowment, 2007.
 - [39] V. Cherkassky and M. Yunqian. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113–126, 2004.
 - [40] P. Cimiano. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer, Berlin, Germany, October 2006.
 - [41] P. Cimiano, G. Ladwig, and S. Staab. Gimme’ the context: context-driven automatic semantic annotation with C-PANKOW. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 332–341, New York, NY, USA, 2005. ACM.
 - [42] P. Cimiano and J. Voelker. Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In A. Montoyo, R. Munoz, and E. Metais, editors, *Proceedings of the Applications of Natural Language to Data Bases (NLDB)*, pages 227–238, Berlin, Germany, 2005. Springer.
 - [43] P. Cimiano and J. Voelker. Towards large-scale, open-domain and ontology-based named entity classification. In G. Angelova, K. Bontcheva, R. Mitkov, and N. Nicolov, editors, *Proceedings of the International Conference on Recent Advances in NLP (RANLP)*, pages 166–172, Borovets, Bulgaria, 2005. INCOMA Ltd.
 - [44] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 89–98, New York, NY, USA, 2004. ACM.

-
- [45] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 2002.
 - [46] J. Curtis, J. Cabral, and D. Baxter. On the application of the cyc ontology to word sense disambiguation. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, pages 652–657, 2006.
 - [47] C. d’Amato, N. Fanizzi, and F. Esposito. Query Answering and Ontology Population: an Inductive Approach. In M. Hauswirth, M. Koubarakis, and S. Bechhofer, editors, *Proceedings of the European Semantic Web Conference (ESWC)*, Lecture Notes in Computer Science, pages 288–302, Berlin, Heidelberg, 2008. Springer.
 - [48] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications ACM*, 5(7):394–397, 1962.
 - [49] G. de Melo, F. M. Suchanek, and A. Pease. Integrating YAGO into the Suggested Upper Merged Ontology. In *Proceedings of IEEE International Conference on Tools with Artificial Intelligence*, Piscataway, NJ, USA, 2008, to appear. IEEE Press.
 - [50] 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.
 - [51] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured web community portals: a top-down, compositional, and incremental approach. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 399–410. VLDB Endowment, 2007.
 - [52] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured web community portals: a top-down, compositional, and incremental approach. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 399–410. VLDB Endowment, 2007.
 - [53] J. Diederich, W.-T. Balke, and U. Thaden. Demonstrating the semantic growbag: automatically creating topic facets for faceteddblp. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 505–505, New York, NY, USA, 2007. ACM.
 - [54] N. K. (Editor). Special issue on data management issues in social sciences. *IEEE Data Engineering Bulletin*, 30(2), 2007.
 - [55] O. Etzioni, M. Banko, and M. J. Cafarella. Machine reading. In *AAAI*, Menlo Park, CA, USA, 2006. AAAI Press.
 - [56] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the International*

- Conference on World Wide Web (WWW)*, pages 100–110, New York, NY, USA, 2004. ACM.
- [57] M. W. Eysenck and M. T. Keane. *Cognitive Psychology: A Student's Handbook*. Psychology Press (UK), 2000.
- [58] U. Feige and M. Goemans. Approximating the value of two power proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the Israel Symposium on the Theory of Computing Systems (ISTCS)*, page 182, Washington, DC, USA, 1995. IEEE Computer Society.
- [59] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
- [60] A. Finn and N. Kushmerick. Multi-level boundary classification for information extraction. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 111–122, Berlin, Germany, 2004. Springer.
- [61] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 577–583, Menlo Park, CA, USA, 2000. AAAI Press.
- [62] W. A. Gale, K. W. Church, and D. Yarowsky. One sense per discourse. In *Proceedings of the workshop on Speech and Natural Language*, pages 233–237, Morristown, NJ, USA, 1992. Association for Computational Linguistics.
- [63] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
- [64] C. Giuliano and A. Gliozzo. Instance based lexical entailment for ontology population. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 248–256. Association for Computational Linguistics, 2007.
- [65] J. Graupmann. Concept-based search on semi-structured data exploiting mined semantic relations. In W. Lindner, M. Mesiti, C. Trker, Y. Tzitzikas, and A. Vakali, editors, *Workshop on Current trends in database technology at EDBT*, pages 31–40, Berlin, Germany, 2004. Springer.
- [66] J. Graupmann, R. Schenkel, and G. Weikum. The SphereSearch engine for unified ranked retrieval of heterogeneous XML and web documents. In K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, editors, *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 529–540, New York, NY, USA, 2005. ACM.
- [67] N. Guarino. Formal Ontology and Information systems. *Formal Ontology in Information Systems*, 1998.

-
- [68] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the International Conference on Computational Linguistics (ICCL)*, pages 539–545. Association for Computational Linguistics, 1992.
- [69] D. Hiemstra and S. Robertson. Relevance feedback for best match term weighting algorithms in information retrieval. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.
- [70] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible SROIQ. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 57–67, Menlo Park, CA, USA, 2006. AAAI Press.
- [71] I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):345–357, October 2004.
- [72] J. C. Hung. Retraction: A new wsd approach using word ontology and concept distribution. *Journal Information Science*, 34(2):231–253, 2008.
- [73] 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.
- [74] G. Ifrim and G. Weikum. Transductive learning for text classification using explicit knowledge models. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proceedings of the European Conference on Principles and Practice of Knowledge Discovery (PKDD)*, volume 4213 of *Lecture Notes in Artificial Intelligence*, pages 223–234, Berlin, Germany, 2006. Springer.
- [75] F. C. J. Iria. Relation extraction for mining the Semantic Web. In *Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
- [76] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the Annual International ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 41–48. ACM Press, 2000.
- [77] R. Jaschke, L. B. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in folksonomies. In *10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, *Lecture Notes in Artificial Intelligence*, pages 506–514, Berlin, Germany, 2007. Springer.
- [78] B. Jaumard and B. Simeone. On the complexity of the maximum satisfiability problem for horn formulas. *Information Processing Letters*, 26(1):1–4, 1987.
- [79] T. Joachims. *Learning to Classify Text Using Support Vector Machines*. PhD thesis, University of Dortmund, Germany, 2002.

-
- [80] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer System Science*, 9(3):256–278, 1974.
 - [81] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, Piscataway, NJ, USA, 2008. IEEE, IEEE Press.
 - [82] G. Kasneci, F. M. Suchanek, M. Ramanath, and G. Weikum. How NAGA uncoils: Searching with entities and relations. In C. L. Williamson, M. E. Zurko, and P. J. Patel-Schneider, Peter F. Shenoy, editors, *Proceedings of the International Conference on World Wide Web (WWW)*, pages 1167–1168, New York, NY, USA, 2007. ACM.
 - [83] M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *Journal of Computer and System Sciences*, pages 382–391. IEEE Computer Society Press, 1990.
 - [84] Z. Kedad and E. Métais. Ontology-based data cleaning. In *Proceedings of the Applications of Natural Language to Data Bases (NLDB)*, pages 137–149, London, UK, 2002. Springer.
 - [85] D. Kinzler. WikiSense - Mining the Wiki. In *Wikimania*, 2005. Available at <http://www.brightbyte.de>.
 - [86] K. Knight. Building a large ontology for machine translation. In *HLT '93: Proceedings of the workshop on Human Language Technology*, pages 185–190, Morristown, NJ, USA, 1993. Association for Computational Linguistics.
 - [87] M. Krötzsch, D. Vrandečić, M. Völkel, H. Haller, and R. Studer. Semantic Wikipedia. volume 5, pages 251–261, Amsterdam, The Netherlands, 2007. Elsevier Science Publishers B. V.
 - [88] S. Liu, F. Liu, C. Yu, and W. Meng. An effective approach to document retrieval via utilizing WordNet and recognizing phrases. In *Proceedings of the Annual International ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 266–272, New York, NY, USA, 2004. ACM.
 - [89] G. Luo, C. Tang, and Y. li Tian. Answering relationship queries on the web. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 561–570, New York, NY, USA, 2007. ACM.
 - [90] A. Maedche and S. Staab. Discovering conceptual relations from text. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, Amsterdam, The Netherlands, 2000. IOS Press.
 - [91] A. Maedche and S. Staab. Learning ontologies for the Semantic Web. In *Proceedings of the Workshop on the Semantic Web at WWW*, 2001.
 - [92] A. Maedche and S. Staab. Ontology learning from text. In *Proceedings of the International Conference on Applications of Natural Language to Information Systems-Revised Papers*, page 364, London, UK, 2001. Springer.

-
- [93] A. Maier, H.-P. Schnurr, and Y. Sure. Ontology-based information integration in the automotive industry. In *Proceedings of the International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 897–912. Springer, 2003.
 - [94] C. D. Manning and H. Schutze. *Foundations of Statistical NLP*. MIT Press, Cambridge, MA, USA, 1999.
 - [95] C. Matuszek, J. Cabral, M. Witbrock, and J. Deoliveira. An introduction to the syntax and content of cyc. In *Proceedings of the AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49, Menlo Park, CA, USA, 2006. AAAI Press.
 - [96] Merriam-Webster. *Merriam-Webster's Collegiate Dictionary*. Merriam-Webster, 2003.
 - [97] D. N. Milne, I. H. Witten, and D. M. Nichols. A knowledge-based search engine powered by Wikipedia. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 445–454, New York, NY, USA, 2007. ACM.
 - [98] R. Montague. Universal grammar. In *Formal Philosophy. Selected Papers of Richard Montague*. Yale University Press, 1974.
 - [99] Network Working Group. Uniform Resource Identifier (URI): Generic Syntax, 2005. <http://tools.ietf.org/html/rfc3986>.
 - [100] Z. Nie, Y. Ma, S. Shi, J.-R. Wen, and W.-Y. Ma. Web object retrieval. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 81–90, New York, NY, USA, 2007. ACM.
 - [101] R. Niedermeier and P. Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36:2000, 2000.
 - [102] I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems*, pages 2–9, New York, NY, USA, 2001. ACM.
 - [103] N. F. Noy, A. Doan, and A. Y. Halevy. Semantic integration. *AI Magazine*, 26(1):7–10, 2005.
 - [104] A. Okumura and E. Hovy. Building Japanese-English dictionary based on ontology for machine translation. In *HLT '94: Proceedings of the workshop on Human Language Technology*, pages 141–146, Morristown, NJ, USA, 1994. Association for Computational Linguistics.
 - [105] P. Pantel and M. Pennacchiotti. Espresso: leveraging generic patterns for automatically harvesting semantic relations. In *Proceedings of the International Conference on Computational Linguistics (ICCL)*, pages 113–120, Morristown, NJ, USA, 2006. Association for Computational Linguistics.
 - [106] P. F. Patel-Schneider and I. Horrocks. OWL 1.1 Web Ontology Language. <http://www.w3.org/Submission/owl11-overview/>.

-
- [107] S. P. Ponzetto and M. Strube. Deriving a Large-Scale Taxonomy from Wikipedia. In *Proceedings of the American National Conference on Artificial Intelligence (AAAI)*, pages 1440–1445, Menlo Park, CA, USA, 2007. AAAI Press.
 - [108] L. Predoiu. Probabilistic information integration and retrieval in the Semantic Web. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, G. Schreiber, and P. Cudr-Mauroux, editors, *Proceedings of the International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 930–934, Berlin, Heidelberg, November 2007. Springer.
 - [109] C. Ramakrishnan, K. Kochut, and A. P. Sheth. A Framework for Schema-Driven Relationship Discovery from Unstructured Text. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *Proceedings of the International Semantic Web Conference (ISWC)*, volume 4273 of *Lecture Notes in Computer Science*, pages 583–596. Springer, 2006.
 - [110] V. Raman, B. Ravikumar, and S. S. Rao. A simplified np-complete maxsat problem. *Information Processing Letters*, 65(1):1–6, 1998.
 - [111] D. Ravichandran and E. Hovy. Learning surface text patterns for a Question Answering system. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 41–47, Morristown, NJ, USA, 2001. Association for Computational Linguistics.
 - [112] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2), 2006.
 - [113] E. Riloff. Automatically Generating Extraction Patterns from Untagged Text. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 2:1044–1049, 1996.
 - [114] E. Rosch, C. Mervis, W. Gray, D. Johnson, and P. Boyes-Bream. Basic objects in natural categories. *Cognitive Psychology*, pages 382–439, 1976.
 - [115] M. Ruiz-Casado, E. Alfonseca, and P. Castells. Automatic Extraction of Semantic Relationships for WordNet by Means of Pattern Learning from Wikipedia. In *Proceedings of the Applications of Natural Language to Data Bases (NLDB)*, volume 3513 of *Lecture Notes in Computer Science*, pages 67–79. Springer, 2005.
 - [116] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice Hall, 2002.
 - [117] H. Saggion, A. Funk, D. Maynard, and K. Bontcheva. Ontology-based information extraction for business intelligence. In *Proceedings of the International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 843–856. Springer, 2007.
 - [118] S. Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 2(1), 2008.

-
- [119] D. L. Schacter. Implicit memory: History and current status. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13(3):501–518, 1987.
 - [120] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1033–1044. VLDB Endowment, 2007.
 - [121] H. U. Simon. General bounds on the number of examples needed for learning probabilistic concepts. In *Proceedings of the Annual Conference on Computational Learning Theory (COLT)*, pages 402–411, New York, NY, USA, 1993. ACM Press.
 - [122] D. Sleator and D. Temperley. Sleator parsing english with a link grammar. *International Workshop on Parsing Technologies*, 1993.
 - [123] R. Snow, D. Jurafsky, and A. Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 1297–1304. MIT Press, Cambridge, MA, 2005.
 - [124] R. Snow, D. Jurafsky, and A. Y. Ng. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of the International Conference on Computational Linguistics (ICCL)*, pages 801–808, Morristown, NJ, USA, 2006. Association for Computational Linguistics.
 - [125] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, pages 233–272, 1999.
 - [126] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. Crystal: Inducing a conceptual dictionary. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1314–1319, 1995.
 - [127] M.-H. Song, S.-Y. Lim, D.-J. Kang, and S.-J. Lee. Automatic classification of web pages based on the concept of domain ontology. *Asia-Pacific Software Engineering Conference*, pages 645–651, 2005.
 - [128] J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole, 2000.
 - [129] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.
 - [130] J. Stoyanovich, S. Bedathur, K. Berberich, and G. Weikum. Entityauthority: Semantically enriched graph-based authority propagation. In *Proceedings of the International Workshop on the Web and Databases (WebDB)*, pages 1–6, Beijing, China, 2007.
 - [131] F. M. Suchanek. Representing Ontological Structures in CLOS, Java and FAST. Bachelor’s Thesis, University of Osnabrueck, Germany, 2003. Available at <http://suchanek.name>.

-
- [132] F. M. Suchanek. Ontological reasoning for natural language understanding. Master's thesis, Saarland University, Germany, 2005.
- [133] F. M. Suchanek, G. Ifrim, and G. Weikum. Combining linguistic and statistical analysis to extract relations from web documents. In T. Eliassi-Rad, L. Ungar, M. Craven, and D. Gunopulos, editors, *ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 712–717, Philadelphia, PA, USA, 2006. ACM.
- [134] F. M. Suchanek, G. Ifrim, and G. Weikum. LEILA: Learning to extract information by linguistic analysis. In P. Buitelaar, P. Cimiano, and B. Loos, editors, *Proceedings of the 2nd Workshop on Ontology Learning and Population (OLP2) at COLING/ACL 2006*, pages 18–25, Sydney, Australia, 2006. Association for Computational Linguistics.
- [135] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge - unifying WordNet and Wikipedia. In C. L. Williamson, M. E. Zurko, and P. J. Patel-Schneider, Peter F. Shenoy, editors, *Proceedings of the International Conference on World Wide Web (WWW)*, pages 697–706, Banff, Canada, 2007. ACM.
- [136] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO - A Large Ontology from Wikipedia and WordNet. *Elsevier Journal of Web Semantics*, 6(3):203–217, September 2008.
- [137] F. M. Suchanek, M. Vojnovic, and D. Gunawardena. Social Tags: Meaning and Suggestions. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, New York, NY, USA, 2008. ACM. To appear.
- [138] H. Tanev and B. Magnini. Weakly supervised approaches for ontology population. In P. Buitelaar and P. Cimiano, editors, *Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, number 167, pages 129–143. IOS Press, 2008.
- [139] M. Theobald, R. Schenkel, and G. Weikum. TopX & XXL at INEX 2005 (ad-hoc track). In N. Fuhr, M. Lalmas, S. Malik, and G. Kazai, editors, *Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005*, volume 3977 of *Lecture Notes in Computer Science*, pages 282–295, Dagstuhl Castle, Germany, 2006. Springer.
- [140] L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal of Computing*, 29(6):2074–2097, 2000.
- [141] M. Vargas-Vera, E. Motta, and J. Domingue. AQUA: An ontology-driven question answering system. In M. T. Maybury, editor, *New Directions in Question Answering*, pages 53–57, Menlo Park, CA, USA, 2003. AAAI Press.
- [142] M. Vojnović, J. Cruise, D. Gunawardena, and P. Marbach. Ranking and suggesting tags in collaborative tagging applications. Technical Report MSR-TR-2007-06, Microsoft Research, Feb. 2007.

-
- [143] G. Wang, Y. Yu, and H. Zhu. PORE: Positive-Only Relation Extraction from Wikipedia Text. In *Proceedings of the International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 580–594. Springer, 2007.
 - [144] T. Wang, Y. Li, K. Bontcheva, H. Cunningham, and J. Wang. Automatic extraction of hierarchical relations from text. In Y. Sure and J. Domingue, editors, *Proceedings of the European Semantic Web Conference (ESWC)*, volume 4011 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 2006.
 - [145] N. Weber and P. Buitelaar. Web-based Ontology Learning with ISOLDE. In *Proceedings of the ISWC Workshop on Web Content Mining with Human Language Technologies*, 2006.
 - [146] D. S. Weld, F. Wu, E. Adar, S. Amershi, J. Fogarty, R. Hoffmann, K. Patel, and M. Skinner. Intelligence in Wikipedia. In D. Fox and C. P. Gomes, editors, *Proceedings of the American National Conference on Artificial Intelligence (AAAI)*, pages 1609–1614, Menlo Park, CA, USA, 2008. AAAI Press.
 - [147] World Wide Web Consortium. RDF Primer (W3C Recommendation 2004-02-10). <http://www.w3.org/TR/rdf-primer/>.
 - [148] World Wide Web Consortium. SPARQL Query Language for RDF (W3C Recommendation 2008-01-15). <http://www.w3.org/TR/rdf-sparql-query/>.
 - [149] World Wide Web Consortium. XML Schema Part 2: Datatypes Second Edition (W3C recommendation 2004-10-28). <http://www.w3.org/TR/xmlschema-2>.
 - [150] F. Wu and D. S. Weld. Autonomously Semantifying Wikipedia. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 41–50, New York, NY, USA, 2007. ACM.
 - [151] F. Wu and D. S. Weld. Automatically refining the Wikipedia infobox ontology. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 635–644, New York, NY, USA, 2008. ACM.
 - [152] Z. Xing and W. Zhang. MaxSolver: an efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(1-2):47–80, 2005.
 - [153] F. Xu and H.-U. Krieger. Integrating shallow and deep NLP for information extraction. In *Proceedings of the International Conference on Recent Advances in NLP (RANLP)*, Borovets, Bulgaria, 9 2003.
 - [154] F. Xu, D. Kurz, J. Piskorski, and S. Schmeier. Term extraction and mining term relations from free-text documents in the financial domain. In *Proceedings of the International Conference on Business Information Systems (BI)*, Poznan, Poland, 2002.

-
- [155] R. Yangarber, W. Lin, and R. Grishman. Unsupervised learning of generalized names. In *Proceedings of the International Conference on Computational Linguistics (ICCL)*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.
 - [156] M. Yannakakis. On the approximation of maximum satisfiability. In *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 1–9, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.
 - [157] E. N. Zalta, editor. *The Stanford Encyclopedia of Philosophy*. The Metaphysics Research Lab, Stanford University, Stanford, California, USA.
 - [158] C. Zhai and J. Lafferty. A risk minimization framework for information retrieval. *Information Processing Management*, 42(1):31–55, 2006.
 - [159] Q. Zhang, F. M. Suchanek, L. Yue, and G. Weikum. TOB: Timely ontologies for business relations. In *Proceedings of the International Workshop on the Web and Databases (WebDB)*, 2008.
 - [160] Z. Zhang. Weakly-supervised relation classification for information extraction. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 581–588, New York, NY, USA, 2004. ACM.
 - [161] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. Simultaneous record detection and attribute labeling in web data extraction. In *ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 494–503, New York, NY, USA, 2006. ACM.
 - [162] J. Zhu, B. Zhang, Z. Nie, J.-R. Wen, and H.-W. Hon. Webpage understanding: an integrated approach. In *ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 903–912, New York, NY, USA, 2007. ACM.
 - [163] C. Zirn, V. Nastase, and M. Strube. Distinguishing between instances and classes in the wikipedia taxonomy. In *Proceedings of the European Semantic Web Conference (ESWC)*, volume 5021 of *Lecture Notes in Computer Science*, pages 376–387, Berlin, Germany, 2008. Springer.