University of Coimbra
Faculty of Sciences and Technology
Department of Informatics Engineering

Probabilistic Reasoning in the Semantic Web using Markov Logic

MSc Thesis

Pedro Carvalho de Oliveira
Candidate

Paulo Jorge de Sousa Gomes
Supervisor

July 2009

Knowledge and Intelligent Systems Laboratory
Cognitive and Media Systems Group
Centre for Informatics and Systems of the University of Coimbra

Abstract

The Semantic Web envisions a world where agents share and transfer structured knowledge in an open and semi-automatic way. In most of the cases, this knowledge is characterized by uncertainty. However, Semantic Web languages do not provide any means of dealing with this uncertainty. They are mainly based on crisp logic, unable of dealing with partial and incomplete knowledge. Reasoning in the Semantic Web resigns to a deterministic process of verifying if statements are true or false.

In the last years, some efforts have been made in representing and reasoning with uncertainty in the Semantic Web. These works are mainly focused on how to extend the logics behind Semantic Web languages to the probabilistic/possibilistic/fuzzy logics, or on how to combine these languages with probabilistic formalisms like Bayesian Networks. In all of these approaches, this is achieved by annotating the ontologies with some kind of uncertainty information about its axioms, using this information to perform uncertainty reasoning. Nevertheless, several questions arise: how can reasoning be done efficiently with this uncertainty information? Where to get this uncertainty information?

In this thesis, we present solutions for both questions. The solution for the first question is Markov logic, a new promising approach to reasoning with uncertainty. In this type of logic, there is no right and wrong world; there are multiple worlds with different degrees of probability. This is done by combining logic and probability in the same representation, and then using efficient learning and inference algorithms.

For the second question, several solutions were developed:
- If the ontologies are annotated with some kind of uncertainty information, like probabilities, Markov logic can be used to reasoning about this information;
- If the ontology contains individuals, those individuals can be used to automatically learn the uncertainty of the ontology;
- If the ontology does not comprise uncertainty information or individuals, both resources can be automatically learned by analyzing textual resources and web search engines.

We developed a system, *Incerto*, which explores the capabilities of Markov logic for the Semantic Web. This system was applied in several interesting tasks, like reasoning about automatically learned ontologies and social networks analysis.

The main contributions of this thesis are:
- The application of Markov logic for learning and reasoning about uncertainty in the Semantic Web;
- The development of several techniques for learning automatically the uncertainty of ontologies;

- The development of a new technique to parameterize Markov logic networks with probabilities;
- The development of a new technique to learn the probability of ontology axioms by using web search engines;
- The development of *Incerto*, and its application to several Semantic Web domains.

Contents

Figures

Tables

1.   Introduction

The Semantic Web (T. Berners-Lee, J. Hendler, and Lassila 2001) envisions a world where agents share and transfer structured knowledge in an open and semi-automatic way. This knowledge, in most of the cases, is characterized by uncertainty (Lindley 2006), i.e., there is a lack of certainty in assigning a true/false value to a certain knowledge statement. However, the Semantic Web does not provide any means of dealing with knowledge uncertainty. Its languages, like RDF and OWL, are mainly based on crisp logic, being incapacitated of dealing with partial and incomplete knowledge. Reasoning in the Semantic Web resigns to a deterministic process of verifying if statements are true or false.

One field that has been trying to tackle the problem of knowledge uncertainty is the field of probabilistic reasoning (Pearl 1988). This field studies methodologies to represent and reason about uncertain knowledge through the use of probability theory. Some of the developed models, like Bayesian and Markov networks (Roller et al. 2007), are considered the state of the art on dealing with uncertainty. In fact, based on the success of probabilistic reasoning and other similar areas, some efforts have been made on representing and reasoning with uncertainty in the Semantic Web (Thomas Lukasiewicz and Umberto Straccia 2008). These works mainly focused on extending the logics behind Semantic Web languages with probabilistic/possibilistic/fuzzy concepts, or on combining these languages with probabilistic formalisms like Bayesian networks. However, most of these approaches have some problems of applicability in real domains, mainly because its complexity and domain restrictiveness.

Recently, a new area of research, called statistical relational learning (SRL) (Lise Getoor and Ben Taskar 2007), has arisen. SRL tries to expand probabilistic reasoning to complex relational domains, like the Semantic Web. This is achieved by combining representation formalisms, like logic and frame-based systems, with probabilistic models. Some of its more expressive and complete approaches, like Bayesian logic (Milch et al. 2007) and Markov logic (Pedro Domingos, Stanley Kok, et al. 2008), have proven to provide interesting capabilities on learning and reasoning about uncertainty in many real world domains.

<u>The objective of this thesis</u> is to study mechanisms to perform probabilistic reasoning in the Semantic Web. For this purpose, we use Markov logic, a novel representation formalism that combines first-order logic with probabilistic graphical models.

In Markov logic, unlike first-order logic, worlds that violate formulas are not impossible, but only less probable. This is achieved by attaching weights to first-order formulas: higher the weight, bigger the difference between a world that satisfies the formula and one that does not, other things being equal. These weighted formulas represent a Markov logic network, which can be seen as a template to construct Markov networks from given sets of constants: each ground atom is a variable, logical connectives are the edges between variables, and each grounded formula is a feature. The resulting Markov network gives a probability

distribution over the possible worlds, being used to answer any probabilistic query about the domain.

Therefore, to apply Markov logic in the Semantic Web, two objects are needed: first-order formulas and their weights. *Formulas* can be acquired by interpreting the semantics of Semantic Web languages as sets of first-order formulas. Since most of these languages represent ontologies based on Description Logics (Baader et al. 2007), they follow a model-theoretic semantics, having a direct correspondence with formulas in first-order logic. *Weights* can be acquired by several ways: if ontologies are annotated with some kind of uncertainty values, like probabilities, we can interpret these values as weights and perform reasoning; if ontologies contain individuals, these individuals can be used to learn the weights; if we do not have uncertainty annotations or individuals, both can be learned through the analysis of textual resources and web search engines.

To demonstrate the feasibility of our approach, we developed *Incerto*, a system that provides a Semantic Web interface to Markov logic reasoning and learning capabilities. This system can be accessed visually and programmatically, and was used in many interesting Semantic Web tasks, like ontology learning (Maedche 2002) and social network analysis (Mika 2007).

The main contributions of this thesis are:
- The application of Markov logic for learning and reasoning about uncertainty in the Semantic Web;
- The development of several techniques for learning automatically the uncertainty of ontologies;
- The development of a new technique to parameterize Markov logic networks with probabilities;
- The development of a new technique to learn the probability of ontology axioms by using web search engines;
- The development of *Incerto*, and its application to several Semantic Web domains.

The organization of this thesis is as follows:
- *Chapter 2* reviews the main concepts that are used in this thesis. A description of the related work is also provided;
- *Chapter 3* describes our proposed approach, with several application examples;
- *Chapter 4* presents the main features of the developed system;
- *Chapter 5* gives general conclusions and future directions of this work.

2. State of the Art

In this chapter, we present the main concepts used in this thesis. First, a review on logic and probabilistic reasoning is provided. Next, we introduce the field of statistical relational learning, followed by an explanation about the main concepts behind the Semantic Web. Finally, an introduction to Markov logic is provided, followed by a description of the most relevant related work to this thesis.

2.1. Logic

Over the centuries, logic has been studied has the underlying mechanism of valid demonstration and inference. The roots of many disciplines, like philosophy, mathematics, and computation, are based on the principles of logic. In this work, logic is used as a *knowledge representation formalism*. For this purpose, logic provides a meaningful and unambiguous way of representing knowledge. The next sections describe three types of logic that are of most importance to this work: propositional logic, first-order logic, and description logics. But before, a slight introduction to the general logic concepts is provided.

2.1.1. Logic as a Knowledge Representation Formalism

The main component of logic as a knowledge representation formalism is the *knowledge base* (KB) (Stuart Russell and Norvig 2002). Each logical KB is composed by a set of *formulas*, or *sentences*, expressed in a logical language. These formulas are expressed according to the syntax of the logical language, i.e., the specification if the formulas are well formed. Their semantics define the truth of each sentence with respect to each *possible world*. For example, the sentence *FatherOf(x,y)* is true in a world where *x* is the father of *y*. These possible worlds are usually called *models*.

Inference, i.e., derive new facts and relations from existing knowledge, is the main objective of a logical KB. Inference algorithms use the *entailment* between sentences, i.e., if a sentence follows logically from another sentence, in this task. Entailment is usually represented by $\alpha \vDash \beta$, meaning that in every model where α is true, β is also true. For example, $FatherOf(x,y) \vDash SonOf(y,x)$ because in every model where *x* is father of *y*, *y* is also the son of *x*. Inference can be defined as the process of finding a specific entailment in a KB. This is usually represented by $KB \vdash_i \alpha$, meaning that the entailment α is derived from *KB* using inference algorithm *i*.

There are two important properties of inference algorithms: soundness and completeness. An algorithm is *sound* if it only derives entailed sentences (i.e., it only derives sentences that are valid with respect to their semantics). An algorithm is *complete* if it can derive any sentence that is entailed (i.e., it can derive all the true sentences).

Since logical KBs provide a representation of knowledge, the correspondence between the representation and the real knowledge must be defined. This procedure is called *grounding*, and is usually made by establishing connections between the symbols and their meanings.

Next, a simple logic to represent logical KBs, propositional logic, is defined.

2.1.2. Propositional Logic

A *propositional logic KB* (Stuart Russell and Norvig 2002) is a set of formulas in propositional logic. Formulas are composed by *symbols*, representing *prepositions* that can be true or false. Symbols are usually represented in uppercase names (e.g., *A* and *BOB* are symbols). There are two symbols with fixed meaning: *TRUE* is the always-true preposition, and *FALSE* is the always-false preposition. These kind of simple formulas, with only one symbol, are called *atomic formulas* (or *atoms*). More complex formulas can be built from atoms by using *logical connectives*.

Connective	Meaning
¬	Not
∧	And
∨	Or
⇒	Implication
⇔	Biconditional

Table 1. Logical connectives.

Logical connectives have an order of precedence ($\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, with decreasing precedence), but delimiters can be also used to ensure precedence (e.g., $A \vee (B \Rightarrow C)$ instead of $A \vee B \Rightarrow C$).

The models of propositional logic are defined by simply assigning true or false values to the proposition symbols (i.e., for *N* propositional symbols, there are 2^N models). The truth value of a formula in a model can be easily computed:
- Atomic formulas already have their trueness computed (they are either true or false);
- Complex formulas are computed recursively using *truth tables*:

P	Q	¬P	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Table 2. Logical truth table.

Inference in propositional logic is both sound and complete, and several algorithms, like *model-checking* and *resolution* (Stuart Russell and Norvig 2002), were studied.

Propositional logic is a simple and powerful language to represent knowledge. However, it lacks of expressive power to describe concisely very large domains. For example, it is difficult to state the fact that all the persons have a name, since we have to write a formula with that information for each person, increasing the number of models exponentially by the number of persons. For this reason, more expressive languages have been developed. First-order logic is one of them.

2.1.3. First-order Logic

First-order logic (FOL) (Barwise and Etchemendy 2002) (Stuart Russell and Norvig 2002) builds a more expressive language in the foundations of propositional logic. This new language is designed to create accurate real world models, characterized by a large number of objects, with relations between them.

A *first-order logic KB* is a set of formulas in first-order logic. FOL uses four types of symbols:

- *Constants*, representing objects (i.e., individuals) in the domain (e.g., *Anna*, *Bob*);
- *Variables*, ranging over constants (e.g., *x*, *y*);
- *Functions*, mapping objects to objects (e.g., *motherOf(Bob)=Anna*);
- *Predicates*, representing binary relations between objects (e.g., *Friends(Anna, Bob)*).

Variables can be *typed*, meaning that they can only model a restricted range of objects (e.g., variable *x* only ranges about *People*).

A *term* is an expression representing an object. It can be a constant, a variable or a function of finite arity (e.g., *x*, *Pedro* and *Friends(x,y)* are terms). A term is grounded when it contains no variables (e.g., *Friends(Anna, Bob)* is a grounding of *Friends(x,y)*).

The simplest FOL formula is an *atom*, composed by a predicate with a finite number of terms as parameters (e.g., *Friends(x,Pedro)* and *Friends(Pedro,motherOf(Bob))* are atoms). Just like propositional logic, more complex formulas can be formed by joining atoms with logical connectives ($\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$) and delimiters. FOL also supports *quantifiers*.

Quantifier	Meaning
$\forall$	Universal Quantification
$\exists$	Existential Quantification

Table 3. First-order logic quantifiers.

A formula is grounded when all of its terms are also grounded. Some examples of formulas are provided:

1. $\forall x \forall y \forall z : Friends(x,y) \wedge Friends(y,z) \Rightarrow Friends(x,z)$
2. $\forall x : Friends(x, MotherOf(x))$
3. $Neighbors(Portugal, Spain)$

Formula 1 states that friends of friends are also friends. In the second formula, everyone is friend of his mother, and formula 3 states that two countries, *Portugal* and *Spain*, are neighbors.

Models in FOL are tuples *(D,I)*, where *D* is a set of domain elements, and *I* is an interpretation. The *domain elements* are the objects contained in that model (e.g., the domain of $\forall x,y \; Friends(x,y)$ is all the objects that can take the values *x* and *y*). The *interpretation* is an assignment that maps objects, functions and relations into symbols (e.g., one possible interpretation of *motherOf(Pedro)* is that *motherOf* refers to the motherhood relation, and *Pedro* refers to the author of this work). Since symbols can have various interpretations, there are distinct models relating to those interpretations. The truth value of a formula can be determined given a model and an interpretation of the values of the variables (i.e., an assignment of objects from the domain to the variables).

Every first-order KB can be translated to *clausal form*, also known as *conjunctive normal form* (Barwise and Etchemendy 2002). A KB in clausal form is a conjunction of clauses, each clause being a disjunction of literals. For example, Formula 1 can be translated as $\neg Friends(x,y) \vee \neg Friends(y,z) \vee Friends(x,z)$. This translation is useful in automated inference.

Propositionalization and *unification* techniques (Stuart Russell and Norvig 2002) can be used in FOL inference, providing both sound and complete inference. However, entailment is only *semidecidable* (i.e., the algorithm return *yes* when the right answer is yes, but it fails to say *no* when the right answer is no). Since this characteristic arise problems in many domains, other logical languages, like Description Logics, have been developed.

2.1.4. Description Logics

Description Logics (DLs) (Baader et al. 2007) are a family of logical languages based on *Semantic Networks* and *Frame Systems*. They are specially designed to model terminological domains. Historically, they were designed as a subset of first-order logic, providing decidable reasoning, while maintaining some of the most important expressivity characteristics of it. DLs use two types of symbols:
- *Atomic concepts*, representing sets of individuals;
- *Atomic roles*, expressing relationships between individuals.

Description Logics KBs are composed by *descriptions*. Elementary descriptions are composed by atomic concepts and atomic roles. More complex descriptions can be composed by using *concept constructors*. The following basic concept constructors are usually used:

Constructor	Meaning
⊤	Universal Concept
⊥	Bottom concept
¬	Atomic Negation
⊓	Logical Intersection
⊔	Logical Union
≡	Logical Equivalence

Table 4. Basic Description Logics concept constructors.

All of these constructors have a similar meaning to those presented in the previous logics. However, more complex constructors can also be used:

- $\forall R.C$, called *value restrictions*, describing individuals which are always connected by role *R* to the concept *C*;
- $\exists R.C$, called *existential quantification*, describing all the individuals connected by role *R* with concept *C*;
- $C \sqsubseteq D$, called *subsumption*, describing that the concept *D* is more general than concept *C*.
- $\leq nR$, called *number restrictions*, restricting the cardinality of the role *R* to the number *n* ($\leq$ can be substituted by $\geq$ or $=$).

DLs divide KBs in two distinct parts: the *intensional* knowledge in the form of a terminology, called *Terminological Box* (TBox), and the *extensional* knowledge, called *Assertional Box* (ABox). The TBox provides the vocabulary, in terms of concepts and rules, of the KB. This is usually done by defining concepts using the logical equivalence constructor (e.g., $Woman \equiv Person \sqcap Female$). The Abox uses the TBox vocabulary to make assertions about individuals. There are two kinds of assertions, corresponding for the two symbols of the language: *concept assertions* (e.g., $Person(PEDRO)$) and *role assertions* (e.g., $Friends(BOB, ANNA)$).

DLs provide a *model-theoretic semantics*. This means that descriptions can be, in most of the cases, identified with formulas in first-order logic. The main idea behind this identification is that concepts correspond to unary predicates, roles to binary predicates, and individuals correspond to constants (e.g., $Woman \equiv Person \sqcap Female$ is interpreted as $\forall x\ Woman(x) \Leftrightarrow Person(x) \wedge Female(x)$).

As previously referred, Description Logics is a family of languages. These languages are characterized by the use of different operators. Usually, all the languages are based in the *Attributive Language* ($\mathcal{AL}$). This language supports atomic negation, concept intersection, universal restrictions, and limited existential

quantification. To distinguish different languages, labels were created to determine the expressivity of these languages.

Label	Meaning
$\mathcal{EL}$	Intersection and Full Existential Restrictions
$\mathcal{F}$	Functional Properties
$\mathcal{E}$	Full Existential Quantification
$\mathcal{U}$	Concept Union
C	Complex Negation
S	ALC with transitive Roles
$\mathcal{H}$	Role Hierarchy
$\mathcal{R}$	Ir/reflexity and role disjointness
O	Nominals
$\mathcal{I}$	Inverse Properties
$\mathcal{N}$	Cardinality Restrictions
$\mathcal{Q}$	Qualified Cardinality Restrictions
$(\mathcal{D})$	Use of Datatype properties

Table 5. Description Logics labels.

For example, the language $\mathcal{SHOIN}$ is $\mathcal{ALC}$ with Transitive Roles (S), plus Role Hierarchy ($\mathcal{H}$), Nominals (O), Inverse Properties ($\mathcal{I}$), and Cardinality Restrictions ($\mathcal{N}$).

The expressivity of the language determines the complexity of its inference. While some DLs are a decidable subset of first-order logic, others can go beyond first-order and have problems of decidability, soundness and completeness. Inference in DLs is usually made by using specific *tableau-based* algorithms (Ralf Moller and Volker Haarslev 2008).

2.2.   Probabilistic Reasoning

Many real world domains are characterized by uncertainty (Lindley 2006). There is only a limited knowledge about the domain, being very difficult to exactly describe the domain or predicting its future behavior. When modeling a domain, uncertainty can be the result of distinct reasons (Stuart Russell and Norvig 2002):
- *Laziness*, when it is too difficult to correctly model the domain;
- *Theoretical ignorance*, when there is not enough information to correctly model the domain;
- *Practical ignorance*, when all the knowledge is available, but the user does not know how to model it correctly.

One way of dealing with uncertainty is through probability theory, where a degree of belief (i.e., a probability) is assigned to the knowledge in the domain. Probabilistic reasoning is an area that tries to find efficient mechanisms to reason under uncertain

knowledge expressed through probability theory. In this area, probabilistic graphical models provide a compact and expressive tool to deal with uncertainty and complexity, by joining concepts from probability theory and graph theory in the same representation. There are two main kinds of probabilistic graphical models: directed (Bayesian networks) and undirected (Markov networks). In this section, the concepts behind these two models are described. First, a brief introduction on probability theory is given.

2.2.1. Probability Theory

The main concept behind *probability theory* (Stuart Russell and Norvig 2002) is the *probability*: a value between 0 and 1 representing the degree of belief in a statement (assigning a probability near 1 to a statement represents a strong belief in that statement, while near 0 represents discredit in the statement).

The basic unit of probability theory is the *random variable*, which represents a variable with initial value unknown. This variable has a domain, representing the values that it can take. In this work, we focus on variable with discrete domains (e.g., the random variable *Weather* can take the values *{sunny, rainy,cloudy,snow}*). Random variables can be joined in *propositions*, using the same logical connectives as logical languages (e.g., $Weather = rainy \land Temperature = low$). Probabilities are given to propositions, declaring the degree of belief in those propositions. There are two main kinds of probabilities:

- *Prior probability P(a)*, which is the degree of belief in preposition *a* in the absence of any other information (e.g., *P(Weather=rainy) = 0.2*);
- *Conditional* (or *posterior*) *probability P(a|b)*, which is the degree of belief in proposition *a* given preposition *b* (e.g., *P(Weather=rainy|Temperature=low) = 0.4*).

Conditional probabilities can be defined in terms of prior probabilities as

$$P(a|b) = \frac{P(a \land b)}{P(b)},$$ **2.1**

$$P(a \land b) = P(a,b) = P(a|b)P(b).$$ **2.2**

Each discrete random variable has a *prior probability distribution* associated. This distribution gives the values for the probabilities of each individual state of the variable (e.g., *P(Weather) = [0.6,0.2,0.15,0.5]* means that *P(Weather=sunny)=0.6*, *P(Weather=rainy)=0.2*, and so on). *Joint probability distributions* between random variables are defined by a combination of states of the variables (e.g., *P(Weather,Temperature)* has a *N* x *M* probability table, being *N* the size of the domain of *Weather*, and *M* the size of the domain of *Temperature*). A joint probability distribution with all the variables in the world is called a *full joint probability distribution*. This type of probability distribution is a complete specification of the uncertainty of the world, and can be therefore used to answer any probabilistic query about the world. For example, the full joint probability distribution table of two binary random variables (*Sunny* and *Hot*) can be defined by:

	Hot	¬Hot
Sunny	0.55	0.1
¬Sunny	0.05	0.3

Table 6. Full joint probability distribution table of binary random variables Sunny and Hot.

Given that distribution, some probabilistic queries can be made:

$$P(Sunny \wedge Hot) = 0.55$$

$$P(Hot) = \ 0.55 + 0.05 = 0.6$$

$$P(Sunny|Hot) = \ \frac{0.55}{0.6} \simeq 0.92$$

Conditional probabilities, like Formula 2.1, can also be defined by the *Bayes' theorem*

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}. \qquad \textbf{2.3}$$

Since the size of the full joint distribution tables grows with the number of random variables (and the size of their domains), this type of probabilistic reasoning can be intractable in many cases (e.g., *N* binary random variables correspond to a table with 2^N elements). For this purposes, more advanced representations were designed to overcome this problem. The field of probabilistic graphical models studies how some interesting properties of graph theory can be used in probabilistic reasoning.

2.2.2. Probabilistic Graphical Models

Probabilistic graphical models (Roller et al. 2007) (Jordan 2004) combine probability theory and graph theory in the same representation, allowing to compactly represent complex domains. The main goal is to efficiently represent a joint probability distribution over a set of random variables. For this purpose, they explore the structure of the distribution, by using the independence properties between random variables, to generate a compact and modular representation of the distribution. The independence property that they explore is the *conditional independence*, i.e., if two random variables are independent in their conditional probability distribution given a third random variable. A random variable *X* is conditionally independent of *Y* given *Z* if

$$P(X, Y|Z) = P(X|Z)P(Y|Z). \qquad \textbf{2.4}$$

There are two main classes of probabilistic graphical models that explore these properties: directed (Bayesian networks) and undirected (Markov networks).

Bayesian Networks

Bayesian networks (Stuart Russell and Norvig 2002) represent a joint distribution of random variables as a directed acyclic graph. Its nodes are the random variables,

while the edges correspond to direct influence from one node to another. Each random variable has an associated *conditional probability distribution*, normally represented as a table (*conditional probability table*, CPT). These CPTs capture the conditional probability of the random variable given its parents in the graph.
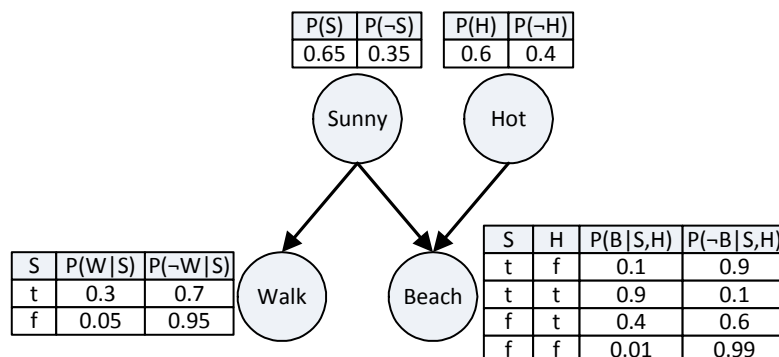
Figure 1. Simple Bayesian network with 4 binary random variables and their conditional probability tables.

This graphical representation provides a complete description of the joint probability distribution of its random variables. This way, the joint probability distribution of a set of random variables $X = \{X_1, \ldots, X_n\}$ can be written as

$$P(X = x) = \sum_{i=1}^{n} P(x_i | parents(X_i)),$$ **2.5**

where $parents(X_i)$ returns the specific values of the parents of the random variable X_i. For example, given the Bayesian network of Figure 1, the probability of being sunny and hot, and we go to the beach but do not take a walk is $P(S = true, H = true, W = false, B = true) = P(S)P(H)P(\neg W|S)P(B|S,H) = 0.65 * 0.6 * 0.7 * 0.9 \simeq 0.25$.

Bayesian networks graph representation also encodes important results to study the interdependence between random variables. The most important is that a node is conditionally independent of its non-descendants given its parents (e.g., *Walk* is conditionally independent of *{Beach,Hot}* given *Sunny*). Other independence assertions can be found by using a complex procedure called *d-separation* (Bishop 2007). This procedure uses the flow of the nodes edges to decide if a set of nodes *X* is independent of another set *Y* given a third set *Z*.

Markov Networks

Markov networks (also called *Markov random fields*) (Roller et al. 2007) (Pedro Domingos, Stanley Kok, et al. 2008) represent a joint probability distribution of random variables as an undirected graph, where the nodes represent the variables, and the edges correspond to some notion of direct probabilistic interaction between neighboring variables. This interaction is parameterized by *potential functions*. There is a potential function for each *clique* (i.e., a completely connected sub-graph) in the graph, being this potential function a non-negative real-valued function of the state of the corresponding clique.

Clique 1		
S	W	Ø(H,B)
t	f	2
t	t	1.7
f	t	0.3
f	f	3.1

Clique 2		
H	B	Ø(H,B)
t	f	1.6
t	t	3
f	t	0.4
f	f	2.5

Clique 3		
B	S	Ø(H,B)
t	f	2.8
t	t	1.7
f	t	0.2
f	f	2.8
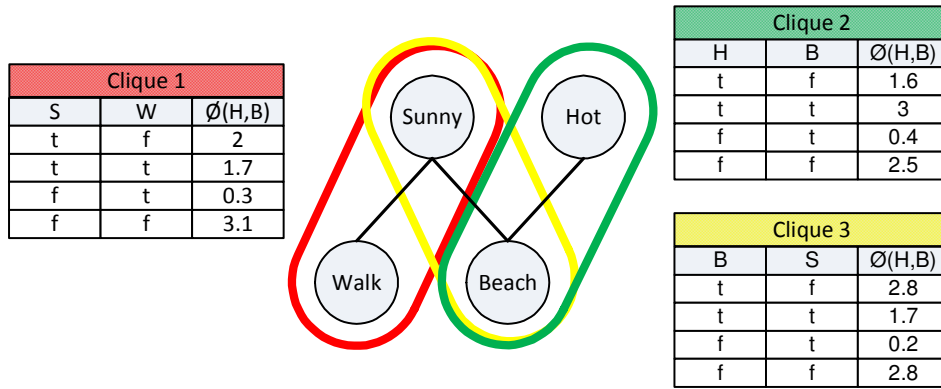
Figure 2. Simple Markov network with 4 binary random variables and their potential functions. Clique 1 is composed by nodes {Sunny, Walk}, Clique 2 by {Hot, Beach}, and Clique 3 by {Beach, Sunny}.

The joint probability distribution of set of random variables $X = \{X_1, \dots, X_n\}$ in a Markov network can be represented by

$$P(X = x) = \frac{1}{Z} \prod_k \emptyset_k(x_{\{k\}}),$$
2.6

where $\emptyset_k$ and $x_{\{k\}}$ are the potential function and the state of the *kth* clique, respectively. This way, the probability of a state can be obtained by multiplying the values of the potential function of all the cliques in that state. Since the sum of all probabilities of all the states it is not 1, it is necessary to divide the formula with a normalizing constant, *Z*, called *partition function*. The partition function is defined by the sum of the product of potentials for all possible states

$$Z = \sum_{x \in X} \prod_k \emptyset_k(x_{\{k\}}).$$
2.7

Just like in Bayesian networks, the probability of being sunny and hot, and we go to the beach but do not take a walk can be easily calculated by using the Markov network of Figure 2: $P(S = true, H = true, W = false, B = true) = \frac{1}{Z}(2 * 1.6 * 2.8) = \frac{8.96}{Z}$.

However, there is a problem with this representation. Since we need a value of the potential function for each state $x_{\{k\}}$ of a clique, the size of the representation is exponential in the size of the cliques. For example, if we have a clique with *N* binary nodes, there are 2^N possible states. So, it is often convenient to use a different way of specifying potentials that can ease this problem. This is done by using *log-linear models*, with each clique potential replaced by an exponentiated weighted sum of features of the state:

$$P(X = x) = \frac{1}{Z} exp\left(\sum_j w_j f_j(x)\right),$$
2.8

$$Z = \sum_{x \in X} exp\left(\sum_j w_j f_j(x)\right).$$
2.9

Now, instead of potential functions, there are features $f_j(x)$, each one with an associated weight w_j. A *feature* is a real-valued function of a state. Each feature can represent more than one state of a clique, being this way a more compact representation than potential functions. If a feature is represented for each state of each clique, the log-linear model can be easily translated to the potential function model, and vice-versa. So, both models can be used interchangeably, depending on the purposes. A simple feature of the previous example can be defined as:
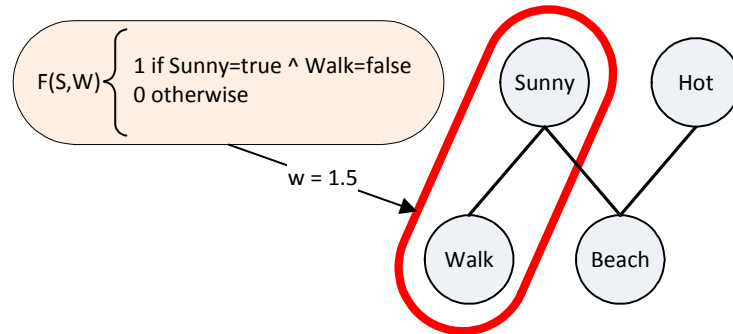
Figure 3. Simple Markov network with 4 binary random variables and one example feature for the clique {Sunny, Walk}.

Like Bayesian networks, Markov networks also encode useful information about the independence between random variables. However, independence in Markov networks is way easier to assert: a node is conditionally independent of the rest of the nodes in the graph given only its immediate neighbors (usually called the *Markov blanket* of a node) (e.g., *Beach* is conditionally independent of *Walk* given *Sunny* and *Hot*).

Comparison between Probabilistic Graphical Models

As seen in the previous sections, both Bayesian networks and Markov networks provide a compact representation of the joint probability distribution over a set of random variables. However, they are both different in many important aspects:

Property	Bayesian Network	Markov Network
Graph	Directed	Undirected
Parameterization	Conditional probabilities	Potential function/Features
Cycles	Not Allowed	Allowed
Partition Function	Not necessary (Z=1)	Necessary
Independence Checking	Parents/D-Separation	Neighbors

Table 7. Comparison between Bayesian networks and Markov networks.

Both approaches have advantages and disadvantages. While Bayesian networks are easier to model and understand, they do not allow cycles and the independence between variables is harder to determine. Markov networks allow cycles and simple independence checking, but its parameterization is hard to understand, and the calculations of the partition function can be very intensive in large domains. So,

when choosing a representation, the domain must be carefully studied to determine the best relation between advantages and disadvantages offered by these representations.

However, for the purpose of the following section, both representations are treated as the same model representation. It is studied that Bayesian networks are a special case of Markov networks, and they can be converted in a Markov network by the construction of a *moral graph*, where the directed edges are transformed in undirected ones, and conditional probability distributions are contained in cliques (for a complete explanation of this process see (Bishop 2007)).

Until now, only the general properties of probabilistic graphical models were provided. In the next section, efficient inference techniques for these representations are demonstrated.

2.2.3. Inference in Probabilistic Graphical Models

Since probabilistic graphical models can represent the full joint probability distribution over the set of random variables $X = \{X_1, \ldots, X_n\}$, they can be used to answer any probabilistic query about the world. There are two main types of queries (Roller et al. 2007):

- The *conditional probability query P(Y | E=e)*, composed by two parts: the *evidence*, a set of random variables *E* and their observed values *e*; and the *query*, a set *Y* of random variables. Our task is to compute the probability distribution over the possible values *y* of *Y*, conditioned on the fact that *E=e*. For example, *P(Beach | Sunny=true, Hot=true)* gives two probability distributions: one when *Beach=true* and another when *Beach=false* ;

- The most probable assignment to some subset of variables given the evidence *E=e*. This type of query has two variants:
  - *Most probable explanation* (MPE), when we want to find the *most likely assignment* to all of the *non-evidence variables*, i.e., if *W=X-E*, and we want to find the most likely assignment to *W* given the fact *E=e*. This can be defined as $argmax_w P(w, e)$. For example, a MPE query given *{Sunny=true, Hot=true}* gives the most likely assignment to the variables *Beach* and *Walk*;
  - *Maximum a posteriori* (MAP), when we want to find the most likely assignment to a set of variable *Y* given the evidence *E=e*, i.e., $argmax_y P(y|e)$. For example, a MAP query of *Beach* given *{Sunny=true, Hot=true}* gives the most likely assignment to the variable *Beach*.

By generating the full joint distribution of the random variables, all of these queries can be answered in a simple way: in a conditional probability query, we just sum the corresponding entries in the joint distribution; in MPE queries we just search for the most likely entries of the non-evidence variables in the joint

distribution; and in the MAP query, we must sum the entries like the conditional probability query and search for the most likely entry. However, generating the full joint distribution is impractical in many cases. So, specialized algorithms were developed to overcome this limitation. There are two main classes of those algorithms: *exact algorithms* and *approximate algorithms*.

Exact Algorithms

If *Z=X-Y-E* is a set of random variables, usually called *hidden variables*, a conditional probability query of *Y* given *E=e* can be calculated as

$$\sum_{Z} P(Y, Z | E = e).$$

2.10

For example, given the Markov network of Figure 2, $P(Beach|Sunny = true, Hot = true) = P(Beach, Walk|Sunny = true, Hot = true)$. However, in the worst case, the complexity of this algorithm, in the case of *n* binary variables, is $O(n2^n)$. The biggest reason of this complexity is that there are many repeated calculations. For this purpose, algorithms based on dynamic programming, like the *variable elimination algorithm* (Stuart Russell and Norvig 2002), can be used. This algorithm evaluates the expression from right-to-left and save the intermediate results, avoiding repeated calculations. In most of the cases, the complexity is reduced to $O(2^n)$. More advanced algorithms, like *join trees* (Jordan 2004), can be used to reduce the complexity to $O(n)$ in the best case. However, even if the complexity of the algorithm is reduced, this problem continues to be *NP-Hard*, requiring exponential time and space to construct the graphical model representation. Approximate algorithms were developed to overcome this limitation.

Approximate Algorithms

The most used approximate inference algorithms are based on *randomized sampling*, being the calculations made in a set of random samples taken from the distribution represented by the graphical model. *Markov Chain Monte Carlo* (MCMC) (Stuart Russell and Norvig 2002) is one of these algorithms.

The algorithm starts with a random state, representing the first sample, and iteratively generates the next samples by sampling the value of one of the non-evidence variables Z_i, conditioned on the current values of the variables in the Markov blanket of Z_i. For example, given the Markov network of Figure 2, if we want to calculate $P(Beach|Sunny = true, Hot = true)$, the first state can be *{Beach=true, Walk=false, Sunny=true, Hot=true}*. In the next state, we can sample *Walk* given its Markov Blanket, i.e., $P(Walk|Sunny = true)$. If the random sample is *Walk=true*, the next state is *{Beach=true, Walk=true, Sunny=true, Hot=true}*. Basically, the algorithm wanders randomly through the search space, flipping the values of the non-evidence variables, maintaining the evidence variables fixed. The visited states are used to estimate the value of the query. For example, if we have 8 states were *Beach=true*

and 2 states where *Beach=false*, $P(Beach = true|Sunny = true, Hot = true) = 0.8$ and $P(Beach = false|Sunny = true, Hot = true) = 0.2$.

2.3. Statistic Relational Learning

Real world data is characterized by high degrees of relational complexity and uncertainty. For example, to assert if a webpage belongs to a certain topic, not only the words in the page have to be taken in account, but also its relations (*hyperlinks*) with another pages. Since webpages can influence each other results, there is a *collective classification* of webpages, where the results for all the webpages are simultaneously decided by using the correlation between pages. Webpages are also characterized by noisy and incomplete information. For example, certain words can be ambiguous (e.g., *Wordnet* (Fellbaum 1998) identifies 18 different senses[1] to the word *bank*), and they can only disambiguated with a certain degree of confidence. So, when dealing with this kind of domains, systems must be prepared to deal with uncertainty.

However, most of the current techniques are not prepared to deal with this type of information. *Statistical relational learning* (SRL) (Lise Getoor and Ben Taskar 2007) is a new area of research that attempts to represent, reason, and learn in domains with complex relational and rich probabilistic structure. SRL tries to combine ideas from the area of *statistical learning*, which deals with uncertainty, and the area of *relational learning*, which deals with complexity, in a unifying representation.

Most of the SRL approaches can be segmented by its *representation formalism*, dealing with the complexity problem, and by the *probabilistic semantics*, dealing with the uncertainty. Representation is usually based on either logic (e.g., first-order logic, logic programs) or frame-based (e.g., entity-relationship models, object-oriented). Probabilistic semantics are mostly based on probabilistic graphical models (e.g., Bayesian networks, Markov networks) or stochastic grammars. Based on this segmentation, the most relevant SRL approaches can be represented in a taxonomy (Figure 4).

[1] http://wordnetweb.princeton.edu/perl/webwn?s=bank
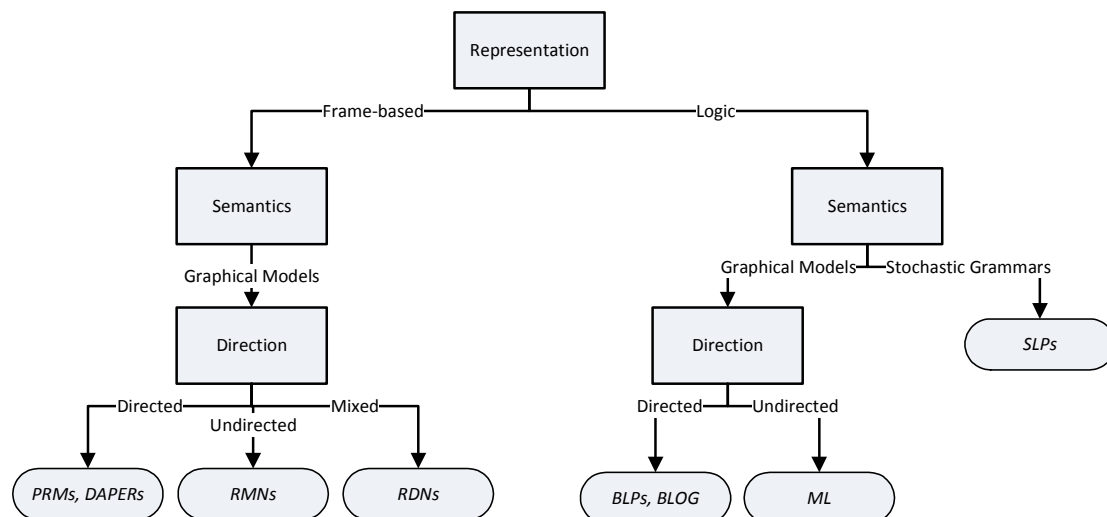
Figure 4. Statistical relational learning taxonomy.

Most of the frame-based systems use probabilistic graphical models as the underlying probabilistic semantic. *Probabilistic Relation Models* (PRMs) (U. Getoor et al. 2007) use pure frame-based systems to extend Bayesian networks with the concepts of objects, their properties, and relations between them. This way, Bayesian networks can naturally deal with relational complexity. Given a database of objects and their relations, a PRM define a probability distribution over the attributes of the objects. PRMs can be manually built or learned from existing databases. *Direct Acyclic Probabilistic Entity-Relationship Models* (DAPERs) (Heckerman, Meek, and Koller 2007) also use Bayesian networks as the probabilistic semantics. However, DAPERs focus on a more expressive frame-based representation than PRMs: *entity-relationship* (ER) models. They extend ER models to handle probabilistic relationships, creating a statistical modeling tool to model probabilistic ERs.

Relational Markov Networks (RMNs) (B. Taskar et al. 2007) combine relational models and Markov networks, providing a probabilistic modeling tool to model relational models. RMNs provide a compact way of representing Markov networks over a relational domain by specifying the cliques and potentials between attributes of related entities. Since an RMN provides a model of the structure of the Markov network, it can be used to provide a coherent distribution over any instance (or collection of instances) that fits in this model. The uncertainty of the model (i.e., its parameterization) can be learned from example data by using *discriminative learning* or *conjugate gradient* methods combined with *belief propagation*.

Relational Dependency Networks (RDNs) (Neville and Jensen 2007) is an extension of dependency networks for relational data. *Dependency networks* are mixed probabilistic graphical models that combine in the same representation both directed and undirected relations between variables. The main difference to Bayesian and Markov networks, apart its mixed representation, is that dependency networks are a purely approximate models (they do not guarantee a coherent

probability distribution). By using *sample-based* techniques, inference and learning can be easily implemented in RDNs.

Logic is another representation used in SRL approaches. *Stochastic Logic Programs* (SLPs) (Muggleton and Pahlavi 2007) combine logic programs with stochastic grammars. *Logic programs*, in their most basic form, are a subset of first-order logic. They provide a simple and powerful language to represent possible theories of a world in a rule-like syntax. Each rule is composed by a head, also called conclusion, followed by a rule body, also called premise. Logic programs are the base of many logic programming environments, like *Prolog* (Sterling and Ehud Shapiro 1994). *Stochastic grammars* are grammars where each production has an associated probability. This way, when deriving, some grammatical derivations are more relevant than another's. Since logic programs are more expressive than stochastic grammars, the main idea behind SLPs is to lift stochastic grammars to the expressive level of logic programs, by combining both approaches. Both parameters and structure can be easily learned from facts or clauses by using *expectation maximization* techniques and *beam-search*, respectively. *Bayesian Logic Programs* (BLPs) (Rersting and De Raedt 2007) also use logic programs as the underlying representation formalism. However, their semantics are provided by Bayesian networks. Just like in PRMs, Bayesian networks are extended with the concepts of objects, their properties, and relations between them. The difference is that BLPs use logic programs as the formalism to provide those features.

Bayesian Logic (BLOG) (Milch et al. 2007) combines first-order logic with Bayesian networks in the same representation. Compared to BLPs, BLOG is way more expressive, making possible to model more complex domains. BLOG is especially designed to deal with *unknown objects*, i.e., when there are no information about some objects and the system must infer that they actually exist. *Markov Logic* (ML) (Pedro Domingos, Stanley Kok, et al. 2008) combines first-order logic with Markov networks. Each first-order formula has an associated weight, representing a *Markov logic network* (MLN). This MLN can be seen as a template to construct Markov networks from given sets of constants: each ground atom is a variable, logical connectives are the edges between variables, and each grounded formula is a feature. The resulting Markov networks give a probability distribution over the possible worlds. Parameters can be learned generatively or discriminatively, and structure can be learned or refined by using bottom-up structure learning algorithms.

2.4. Semantic Web

The Semantic Web envisions a world where agents share and transfer structured knowledge in an open and semi-automatic way. In this section, the main concepts and technologies behind that vision are described. First, the main reasons behind the need for a Semantic Web are identified, followed by an explanation of its layered architecture.

2.4.1. From the Syntactic Web to the Semantic Web

In the current *World Wide Web (*WWW) most of the information is represented in *Hyper Text Markup Language* (HTML)[2]. This language was made to represent visually the information to the user, and no efforts have been made to make it understandable by machines. For example, consider this excerpt of a simple HTML web page and its Web-browser visualization:

```
1.    <h1>Pedro Oliveira</h1>
2.    <p>MSc Student in Informatics Engineering</p>
3.    <p>University of Coimbra, Portugal</p>
```

Pedro Oliveira

MSc Student in Informatics Engineering

University of Coimbra, Portugal
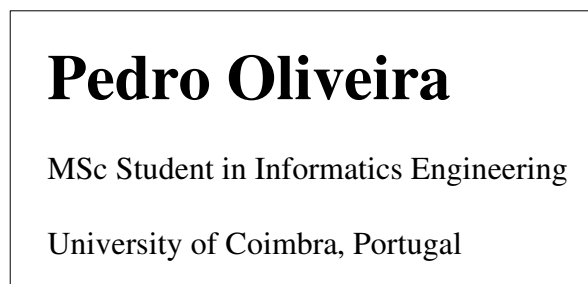
Figure 5. Visual representation of the previous HTML.

By seeing the page, the user understand that this page is dedicated to a person, named *Pedro Oliveira*, a *MSc* student in *Informatics Engineering* at the *University of Coimbra* in Portugal. But to the machine, the only information that it can understand is that there is a header that says *Pedro Oliveira*, with two more paragraphs. But what is *Pedro Oliveira*? Is it a person? A place? Or is just some random meaningless words that the web developer chooses to give to the header of the page? And what is the relation of the header with the proceeding paragraphs?

This web is usually called the *Syntactic Web* (Breitman, Casanova, and Truszkowski 2006), where information presentation is carried out by computers, and the interpretation and identification of relevant information is delegated to human beings. There is no meaning associated with the text, being the markup metadata (e.g., *<h1>*) only used to visually represent the related text. For the machine it is only possible to know the *syntactic* information of the text, i.e., if it is a verb, a pronoun, a number, etc. There is no information about the meaning, i.e., the *semantics*, of the text.

---

[2] http://www.w3.org/html/

In the last years, the scientific community put a lot of effort on trying to extract meaning from unstructured data, like those present in the WWW. Some fields like *Information Retrieval* (Baeza-Yates and Ribeiro-Neto 1999), *Machine Learning* (Bishop 2007), and *Natural Language Processing* (Jurafsky and Martin 2008) have produced increasly complex systems for this task. Some of these systems are the base of very large companies like *Google*[3], *Yahoo*[4], and *SAS*[5]. Although, in all of those systems there is still currently a *knowledge gap* (Mika 2007) between what the machine understands and is able to work with, and what the user knows and infers about the data. This handicap from the machine is mainly consequence of technological difficulties in understanding the contents of a webpage. Since the contents are expressed in natural language, images, videos, or other complex representations, it is very difficult to the machine recognize the meaning of those elements.

Most of the times this handicap comes from the lack of background knowledge of the machine. In the previous example, we know that if something has a name and has some information about a scholar position, it is probable that this scholar position is of the person referred by the name. Although it is easy to make this assertion by a person, this task is difficult for a machine. Even if the machine can identify that a portion of text is a name or a scholar position, how can it determine that a scholar position is normally related to a person if that information is not stated anywhere? This is where the Semantic Web enters.

Envisioned in 1998 (Tim Berners-Lee 1998) by Tim Berners-Lee, the inventor of the WWW and HTML, the *Semantic Web* tries to fill the knowledge gap between human and machine. The main objective of the Semantic Web (T. Berners-Lee, J. Hendler, and Lassila 2001) is to "bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users." With that meaningful knowledge, software agents while reading a page can not only assert the keywords and links of the document, like today search engines, but also complex relations between the elements of the page, just like those described in the later example.

This Semantic Web is not intended to be apart from the current Web. There is no effort on making a new Web, where all the information is somehow more intelligent and structured than it is today. The Semantic Web is not a separable Web, but an extension to the current one. In the future Web, there will coexist "normal" Web

---

[3] http://www.google.com/

[4] http://www.yahoo.com/

[5] http://www.sas.com/

Pages and Semantic Web pages, with the only difference being the later ones more complete to the desires of the user. This way, Semantic Web is trying to expand a web of documents for people with a web of information for machines.

To realize this vision it is necessary to formalize technologies, tools, and standards that provide the ability to incorporate meaningful information onto Web pages. Those technologies need to express both data and rules for reasoning about data, allowing machines to execute complex tasks and better cooperate with humans. In the next sections, these technologies are described.

2.4.2. Semantic Web Architecture

As stated before, a set of technologies, tools, and standards is needed to realize the Semantic Web vision. A *Layered Architecture* (Tim Berners-Lee 2000), composed by a series of standards structurally organized, was proposed (Figure 6). In this architecture, interrelationships of growing complexity between standards are structurally represented.
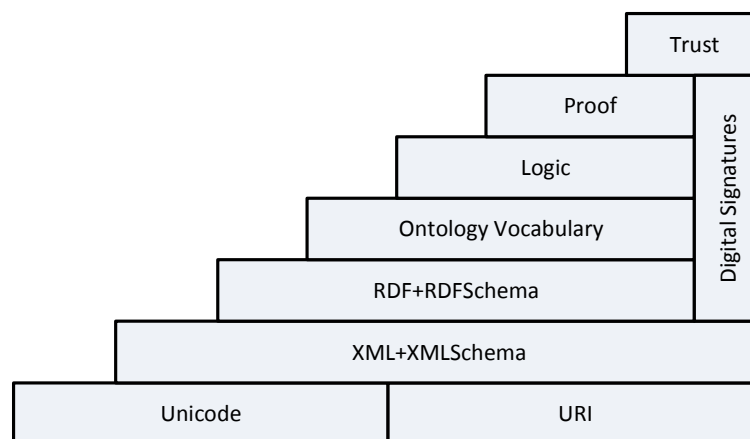
Figure 6. Semantic Web layered architecture (adapted from (Tim Berners-Lee 2000)).

Its foundation is composed by *Unicode* and *URI*, both responsible for the identification of resources. Above, there are 3 layers of representation: *XML+XMLSchema* that provides a syntactic operability layer, letting users write structured Web documents with a user-defined vocabulary; the *RDF+RDFSchema* layer defines a basic data model to write statements about resources using XML; and the *Ontology Vocabulary*, composed of a more complex representation of knowledge. At the top, the most complex layers: *Logic*, *Proof*, and *Trust*. Some layers rely on *Digital Signatures* to ensure security. In the next sections, all of these layers are explained.

2.4.3. URI and Unicode

Before explaining URIs and Unicode, it is useful to define the most basic unit in the Semantic Web: the resource. A *resource* (Breitman, Casanova, and Truszkowski 2006) is "anything that has an identity, be it a retrievable digital entity, a physical entity, or

a collection of other resources". All the information in the Semantic Web is represented as resources. So, it is necessary to define a language to refer to those resources.

A *Universal Resource Identifier* (URI) (Berners-Lee 2005) is a formatted string that globally identifies an abstract or physical resource. The most used type of URI is the *Universal Resource Locater* (*URL),* which identifies resources via an abstract identification of the resource location (e.g., *http://student.dei.uc.pt/pcoliv*).

URIs can have a *fragment identifier* attached, preceded by a *#*, meaning that a resource is subordinated to another, primary resource. In the Semantic Web these are normally used to identify resources in the same namespace, and are usually called *URIRefs* (URI References) or *Hash URIs* (Sauermann and Cyganiak 2008). For example, *http://student.dei.uc.pt/pcoliv#a* and *http://student.dei.uc.pt/pcoliv#b* means that that resources *a* and *b* are in the same namespace, *http://student.dei.uc.pt/pcoliv*.

URIs can be used in two ways: by using the *absolute URI* that identifies a resource independently of the context in which the URI appears (e.g., *http://student.dei.uc.pt/pcoliv#a*); or by using a *relative URI* with some prefix, previously declared, omitted (e.g., *pcoliv#*a).

To promote interoperability between knowledge sources in the Semantic Web it is necessary that all the parts involved use the same encoding to refer to resources. This is done by using *Unicode* (The Unicode Consortium 2006), an encoding system that provides a unique number for every character, independently of the platform, program, or language.

2.4.4. XML

The *Extensible Markup Language* (XML) (Antoniou and Harmelen 2008) is a general-purpose markup language, designed to describe structured documents. In contrast to HTML, users can construct their own tags in XML (e.g., *<scholar_position>*). XML does not provide semantics indicating how to visualize documents. It is a language more suitable to machines, being normally used in data integration, interoperability, and exchange tasks (Cardoso 2007).

A XML document is composed of plain text and markups (in the form of *tags*). The example from Section 2.4.1 can be easily represented in XML:

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.  <page>
3.  <person personid="http://student.dei.uc.pt/pcoliv">
4.  <name>Pedro Oliveira</name>
5.  <scholar_position>MSc Student in Informatics Engineering</scholar_position>
6.  <address>University of Coimbra, Portugal</address>
7.  </person>
8.  </page>
```

A XML document usually starts with a *preprocessing* instruction (line 1) defining some general parameters like the XML version and encoding, being followed by a set of *elements* consisted by a *start tag* (e.g., *<name>*), the *content* (e.g., Pedro Oliveira) and an *end tag* (e.g., *</name>*). Elements can be nested, like *<name>* is nested in *<person>*, being a valid XML document a balanced tree of nested elements (Figure 7). An element can have one or more name-value pair attributes inside the opening tag, like *personid="http://student.dei.uc.pt/pcoliv"* in *<person>*.
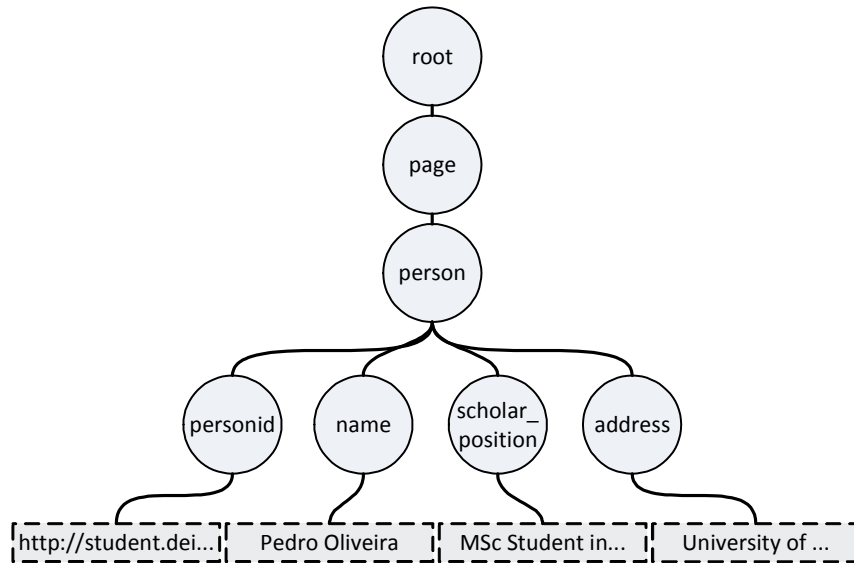
Figure 7. XML document tree of the previous XML example.

To be considered a well-formed XML document, the document must satisfy some syntactic constraints defined in a formal grammar, such as every start tag must have an end tag, attributes within an element have unique names, among others (Antoniou and Harmelen 2008). Besides those restrictions, users can impose its own restrictions in the structure of the document using a *Document Type Definition* (DTD) (Antoniou and Harmelen 2008).

A DTD is a XML-based language used to design constraints on the construction of a XML document. Using DTD, users can define all possible elements, their attributes, and even the allowed structure of a XML document. A simple DTD of the previous example can be easily defined:

```
1.    <!ELEMENT page (person+)>
2.    <!ATTLIST person
3.    personid CDATA #REQUIRED
4.    >
5.    <!ELEMENT person (name)>
```

In this DTD, each page can have one or more persons (line 1). Each person must have an attribute *personid* (lines 2-4), and one *name* element (line 5).

XMLSchema (Antoniou and Harmelen 2008) offers a significantly richer language for defining the structure of XML documents. The main innovation comparing to DTD

is that XMLSchema uses a pure XML language to express the constraints on the structure. XMLSchema support the same restrictions of DTD, adding some other features like support for basic data types, constraints on attributes, sophisticated structures and the use of namespaces to allow the combination of multiple schemas. The same restrictions of the previous DTD can now be stated in XMLSchema:

```
1.   <xsd:schema
2.   xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
3.   version="1.0">
4.   <xsd:complexType name="page">
5.   <xsd:element name="person" minOccurs="1">
6.   </xsd:complexType>
7.   <xsd:complexType name="person">
8.   <xsd:attribute name="personid" type="string">
9.   <xsd:element name="name">
10.  </xsd:complexType>
11.  </xsd:schema>
```

2.4.5. RDF

In the last section, we have seen that XML is a very good choice to express knowledge in a structured way. It provides a general framework, with a consistent syntax, for interchange of data and metadata between applications. However, XML does not provide any way of adding meaning (i.e., semantics) to the data. By nesting elements and defining properties, meaning is not being associated to those elements. They are just being structured; each application decides how to interpret those elements.

Resource Description Framework (RDF) (Breitman, Casanova, and Truszkowski 2006) (Antoniou and Harmelen 2008) is a data model proposed to fulfill the need of giving meaning to XML structured information. An RDF document is composed by a set of statements. Each statement is a triple composed of:

- *Subject*, the resource which the statement refers;
- *Predicate*, also called property, describing relations between resources;
- *Object*, which represents the value of the property, and can be a resource or an atomic value called *literal*.

There are two common ways of textually representing statements: *triples* of the form *(Subject, Predicate, Object)* or in a functional style with binary predicates *Predicate(Subject, Object)*. Both are used interchangeably in this work. Visually, a statement can be seen as a *directed graph* (DAG).
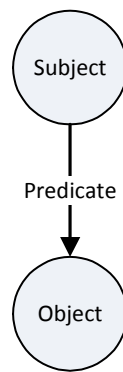
Figure 8. Visual representation of a RDF statement.

Since a RDF document is a set of statements, visually it is also a DAG. For example, the DAG of three statements modeling the domain presented in Section 2.4.1 can be seen on Figure 9.

1. Name(http://student.dei.uc.pt/pcoliv#me, "Pedro Oliveira")
2. Course(http://student.dei.uc.pt/pcoliv#me, http://www.dei.uc.pt/courses#msc_ie)
3. University(http://www.dei.uc.pt/courses#msc_ie, http://www.uc.pt/univ#us)
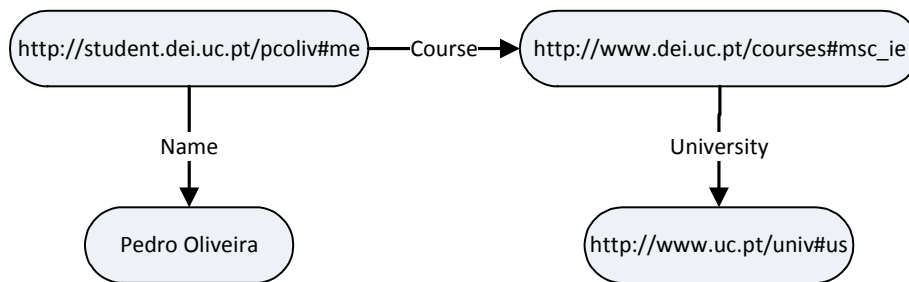
Figure 9. Visual representation of three RDF statements.

Although graphs and triples/functions are useful representations to human understanding, they are not so convenient to the storage, retrieval, and exchange of documents between machines. So, usually, RDF documents are stored in XML:

```
1.   <!DOCTYPE rdf:RDF [
2.   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
3.   ]>
4.   <rdf:RDF
5.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6.   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
7.   xmlns:uni="http://www.university.org/uni-ns#">
8.   <rdf:Description rdf:about="http://student.dei.uc.pt/pcoliv#me">
9.   <uni:Name>Pedro Oliveira</uni:Name>
10.  <uni:Course rdf:resource="http://www.dei.uc.pt/courses#msc_ie"/>
11.  </rdf:Description>
12.  <rdf:Description rdf:about="http://www.dei.uc.pt/courses#msc_ie">
13.  <uni:University rdf:resource="http://www.uc.pt/univ#us"/>
14.  </rdf:Description>
15.  </rdf:RDF>
```

First, *document type* (lines 1-3) and available *namespaces* (lines 4-7) are declared. Resources belonging to declared namespaces can be identified by a relative URI (e.g., *rdf:Description* instead of *http://www.w3.org/1999/02/22-rdf-syntax-ns#Description*). Statements are described in *rdf:Description* elements, using *rdf:about* to define the *Resource*, and representing the *Predicate* and *Object* as XML tags (lines 9-12 and 14-16). To refer an object that is also a resource (i.e., it is not a literal), we use the property *rdf:resource* (lines 11 and 15).

RDF is a very flexible data model, letting users describe resources using their own vocabulary. It does not provide any means for defining domain specific classes and properties. The only defined property is *rdf:type*, which defines the type of any object in the domain. Therefore, a way is needed to define the possible terms used to specify the resources, properties and values referred on RDF statements, and the way resources can be related to each other.

RDF Schema (RDFS) (Antoniou and Harmelen 2008) is an extension to RDF providing a vocabulary to specify classes (*rdfs:class*) and their relations (*rdfs:subClassOf*), properties (*rdfs:property*) and their relations (*rdfs:subPropertyOf*), literals (*rdfs:literal*), annotations (*rdfs:comment* and *rdfs:label*), property restrictions (*rdfs:range* and *rdfs:domain*), among others. This way, users can define classes, individuals and properties in a simple hierarchical structure, using a global agreed vocabulary.

RDFS is fully compatible with RDF, being a well formed RDFS document also a valid RDF document. So, RDFS can also be represented in XML in the same way as RDF.

2.4.6. Ontology Vocabulary

The expressivity of RDF and RDFS is very limited. RDF only provides a simple vocabulary to express structured knowledge, while RDFS is limited to model hierarchies with simple restricted properties. However, more expressivity is needed to model some complex domains. One of the ways to provide this expressivity is by using ontologies.

An *ontology* can be easily defined (James Hendler 2001) as a set of knowledge terms for some particular topic, including the vocabulary, semantic interconnections and rules of logic/inference of those terms. In general, ontologies provide a shared and common understanding of a domain that can be communicated between people and/or machines (Wahlster, Lieberman, and James Hendler 2002). They are normally used to make explicit conceptualizations that describe the semantics of the data with the intuition of reasoning about that data.

The most prominent markup language proposed by the *W3C* to model ontologies in the Semantic Web is the *Web Ontology Language* (OWL) (Breitman, Casanova, and Truszkowski 2006) (Antoniou and Harmelen 2008). OWL, just like RDF and RDFS, is defined as a vocabulary, but with stronger semantics than its predecessors. Some of the new relations are, for example, the inclusion of special properties

(*owl:TransitiveProperty*, *owl:SymetricProperty*, *owl:FunctionalProperty*, and *owl:InverseFunctionalProperty*), enumerations (*owl:oneOf*), versioning information (*owl:versionInfo*), logical properties (*owl:intersectionOf*, *owl:sameAs*, and *owl:differentFrom*), among others.

However, more expressivity means more complexity. So, there are currently three sublanguages in OWL, each one with a different complexity:

- *OWL Full* is OWL with no restrictions. It is fully compatible with RDF/S, both semantically and syntactically. The problem is that the language makes the reasoning over it *undecidable* (i.e., some statements cannot be shown to be either true or false), making almost impossible a complete (or efficient) reasoning.

- *OWL DL* is a sublanguage of OWL that tries to restrict it to the well known Description Logic $\mathcal{SHOIN(D)}$ (Baader et al. 2007). This logic is known of being both decidable (all computations will finish in finite time) and complete (all conclusions are guaranteed to be computable), making its reasoning possible, at least in theory. Those restrictions are made by disallowing the applications of constructors between classes. The problem with this sublanguage is that it is not fully compatible with RDF. The applied restrictions force to make some changes in a valid RDF document to be considered a legal OWL DL document. However, a legal OWL DL document is also a legal RDF Document.

- *OWL Lite* is an even more restricted version than OWL DL, corresponding to the Description Logic $\mathcal{SHIF(D)}$. By excluding enumerated classes, disjoint statements and arbitrary cardinality, a more easy (for both human and machine) sublanguage is created. Although, the expressivity is largely reduced compared to the other OWL sublanguages.

These three sublanguages create an internal sub layer in the Ontology Vocabulary layer in the Semantic Web Architecture (Figure 10). This sub layer shows an upward compatibility (e.g., an OWL Lite ontology is also a valid OWL DL one) of growing complexity.

Figure 10. OWL sub layer.

More recently, a new extension to OWL, called *OWL2* (Grau et al. 2008) has been proposed. This new language extends OWL DL with a new set of features that have

been requested by users. Extra syntactic constructors like *owl:DisjointUnion* and *owl:DisjointClasses*, new constructors for properties (e.g., *owl:ReflexiveProperty* and *owl:AssymetricProperty*), extended datatypes capabilities, and new annotating features are some of the new features. Its Description Logic, $\mathcal{SROIQ(D)}$, is also sound and complete.

2.4.7. Logic

One of the most powerful capabilities of the Semantic Web is *reasoning*, i.e., derive new facts and relations from existing knowledge. The main mechanism behind this reasoning is *logic* (Section 2.1). In fact, the biggest advance made by the Semantic Web is to add logic to the current Web, providing agents with the capability of using rules to make inference, choose courses of action and answer questions.

Take for example the three RDF statements of Figure 9. An agent confronted with this knowledge will infer that *Pedro Oliveira* is connected to the *University of Coimbra* through his *MSc* course, i.e., the *MSc* course serves as a transitive path between *Pedro Oliveira* and the *University of Coimbra*. So, when asked about individuals in the *University of Coimbra*, the agent will return *Pedro Oliveira*, even if that information is not explicitly represented.

As seen in the previous section, the semantics of some OWL languages are based on Description Logics (see Section 2.1.4), a well studied group of logical languages. Even if OWL Full does not have a predefined logic, it can be interpreted as belonging to the set of *high-order languages* (S. Shapiro 2001), which are very expressive logical languages that usually do not provide either sound or complete reasoning. These OWL languages allow us to understand one of the most important conclusions of logic as a knowledge representation mechanism: the difficult of reasoning increases with the expressive power (Levesque and Brachman 1985). Since there is a tradeoff between the expressiveness of the representation language and its computational tractability, there is an increasing need in finding logical languages that are expressive enough, but remain with some interesting capabilities like soundness and completeness. The Semantic Web community has been extensively researching on this theme, and the new language, OWL2 (Grau et al. 2008), is one of these attempts.

2.4.8. Digital Signatures, Proof and Trust

As stated in the previous section, reasoning is one of the most important capabilities of the Semantic Web. However, different reasoners can produce different results. Since in a fully distributed Semantic Web most of the time the reasoning could be made in a different machine than ours, for example using a *Semantic Web Service* (SWS) (Cardoso 2007), how can agents trust that the inference process was reliable and the result is what they were expecting? For example, if there is a SWS that gives the address of a person given its name, how can an agent be sure that the returned

address is from the people that he want and not from another person with the same name?

This can be done by accompanying the results with a justification that allows the receiving agent to assert the quality of the results. This is the purpose of the *Proof layer*. One of the proposed languages to describe justification of results in the Semantic Web is the *Proof Markup Language* (PML) (Silva, McGuinness, and Fikes 2006), using OWL to exchange proofs between machines.

Trust is the top layer of the Semantic Web architecture. Since anyone can contribute to the Semantic Web, some form of trustworthiness must be given to the information. This can be achieved by using *Digital Signatures* (e.g., public key cryptography), trusted agent recommendations, and ratings from certificated agencies. This way a *Web of Trust* (Matthew Richardson, Agrawal, and Pedro Domingos 2003) could be created, were agents can be aware of the credibility and reliability of statements, presenting the most relevant information to the user.

2.5. Markov Logic

Handling uncertainty and complexity in the same representation has been a long goal of Artificial Intelligence. Markov logic is a novel language that provides that capability, joining in the same representation probabilistic graphic models (Markov networks) to handle uncertainty, and first-order logic to handle complexity. By attaching weights to first-order logic formulas, a Markov logic network can be built. This network can be used as a template to construct Markov networks, providing the full expressiveness of probabilistic graphical models and first-order logic.

In the next sections, Markov logic representation and capabilities are described.

2.5.1. Markov Logic Networks

Markov logic (M. Richardson and P. Domingos 2006) (Pedro Domingos, Stanley Kok, et al. 2008) combines first-order logic and Markov networks in the same representation. The main idea behind Markov logic is that, unlike first-order logic, a world that violates a formula is not invalid, but less probable. This is done by attaching weights to first-order logic formulas: the higher the weight, the bigger is the difference between a world that satisfies the formula and one that does not, other things been equal. Using these weighted formulas, a Markov logic network can be built.

A *Markov logic network* (MLN) (M. Richardson and P. Domingos 2006) L is a set of pairs (F_i, w_i) where F_i is a formula in first-order logic and w_i is a real value representing the weight of the formula. If a set of constants $C = \{c_1, \dots, c_n\}$ is provided, a Markov network $M_{L,C}$ can be defined as follows:
- A binary node is created for each possible grounding of each atom in L, being its value 1 if the ground atom is true, 0 otherwise.

- Each possible grounding of each formula F_i in L generates a distinct feature, being its value 1 if the ground formula is true, 0 otherwise. The weight of the feature is the w_i associated with the formula.

This way, it is created a node for each ground atom and an edge if two ground atoms appear in the same formula. Suppose a simple MLN with two formulas.

Formula	Weight
$\forall x : Steal(x) \Rightarrow Prison(x)$	3
$\forall x \forall y : CrimePartners(x, y) \wedge Steal(x) \Rightarrow Prison(y)$	1.5

Table 8. Markov logic network example.

Using the previous algorithm, if we have two constants, *Anna* and *Bob*, the resulting Markov network has eight variables, corresponding to eight grounded atoms.
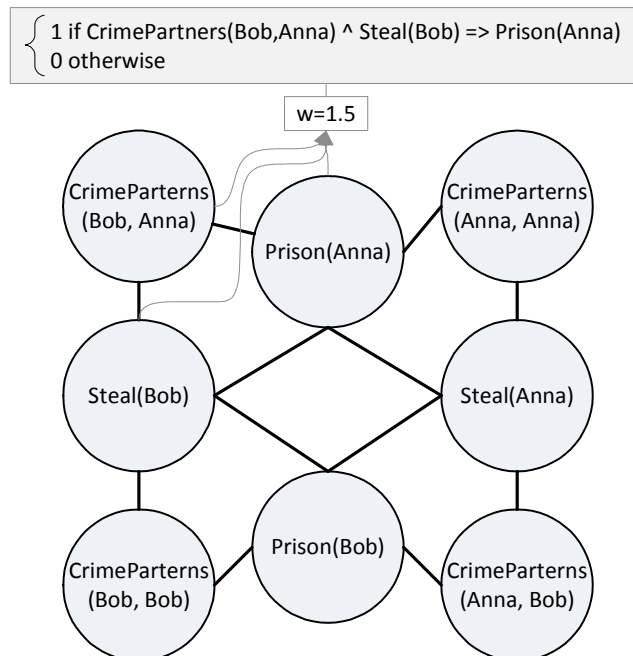
Figure 11. Markov network, with an example feature, built from the previous Markov logic network.

An MLN can be viewed as a template to construct Markov networks. With different sets of constants, different Markov networks are created. However, all of those networks have some kind of similarity in the structure or in the parameters. For example, all the groundings of a formula have the same weight, making the Markov networks of the different groundings of the same formula similar in structure. These grounded networks are called *grounded Markov networks*.

The probability distribution of a ground Markov network can be defined as

$$P(X = x) = \frac{1}{Z} exp\left(\sum_{i=1}^{F} w_i n_i(x)\right),$$ **2.11**

where F is the number of formulas in the MLN, $n_i(x)$ is the number of true groundings of F_i in the world x, and w_i is the weight of F_i.

Markov logic can easily represent many of the models usually used in Artificial Intelligence. For example, an MLN with only grounded formulas is a Markov network, and an MLN with all the weights equal and tending to infinity is a first-order logic knowledge base. Other models that use products of potentials, like Bayesian networks, can also be represented in Markov logic (M. Richardson 2004).

There are two main tasks that can be done by MLNs: inference and learning.

2.5.2. Inference

Since MLNs create Markov networks, the inference algorithms presented in Section 2.2.3 can be used in Markov logic. However, given the specific properties of MLNs, more efficient algorithms were developed. *Most probable explanation* (MPE) and *maximum a posteriori* (MAP) queries reduce to find the truth assignment that maximizes the sum of weights of satisfied clauses. This can be done using *MaxWalkSAT* (Selman, Kautz, and Cohen 1993), a local-search weighted satisfiability solver algorithm. However, this algorithm consumes many resources in sparse domains, since it requires the full grounding of the MLN. Lazy versions of MaxWalkSAT, like *LazySAT* (P. Singla and P. Domingos 2006a), solve this problem by gradually grounding the MLN when its distinct parts are needed.

Another inference task is to find the marginal and conditional probabilities of a formula given an MLN and possibly other formulas as evidence. Since a probability of a formula is the sum of the probabilities of the world where it holds, the more ground atoms exist, the more difficult is the task. Approximate inference algorithms, like the previous referred *Markov Chain Monte Carlo* (MCMC) (Section 2.2.3), are usually used. However, MCMC is not efficient in domains where formulas with deterministic or near-deterministic dependencies exist (e.g., formulas with infinite weight) because these areas of the search space can be very difficult to traverse by simple flipping the value of the non-evidence variables. To solve this problem, we can use *MC-SAT* (H. Poon and P. Domingos 2006), a combination of MCMC and the *SampleSAT* satisfiability solver (Wei, Erenrich, and Selman 2004). MC-SAT uses *slice sampling* to help capturing the dependencies between variables, allowing jumping from these difficult areas.

2.5.3. Learning

There are two things that can be learned in a MLN: *parameters* (i.e., the weights) and *structure* (i.e., the features). Both types are learned from example data.

Weights can be learned *generatively* by maximizing the *pseudo-likelihood* (Besag 1975) of the data. This is done by adjusting the weight of each variable by the likelihood of the variable and his neighbors with the given data. If the model predicts that a feature is true less often than it really is in the data, then the weight is increased; otherwise, it is decreased. Since this procedure is made taking only in account the variable and his neighbors, sometimes weights can be overestimated. A solution when it is known a priory which atoms will be evidence and query is to use *discriminative learning* (P. Singla and P. Domingos 2005). It uses a similar approach to the former, but only takes in account the evidence and query atoms to do the learning, maximizing the conditional likelihood of the query predicates.

Features can be learned from an empty knowledge base (KB) or from an existing KB (making possible to refine an existing KB). The idea is to add all the single atoms to the MLN and iteratively try to joining them until finding an MLN that maximizes a certain evaluation metric based on the provided data. This evaluation can be done by using a weighted version of the already defined pseudo-likelihood algorithm. Since this is a very intensive task, some improvements (S. Kok and P. Domingos 2005) are made to reduce the number of calculations, normally using approximation techniques that avoid the update of all the weights in each iteration. Instead of randomly joining atoms, relationship graphs between variables can be used to guide the learning (Mihalkova and Mooney 2007).

2.6. Related Work

This section is dedicated to related work in the field of reasoning in the Semantic Web. There are three main research themes relevant to this topic: *deterministic reasoning in the Semantic Web*, the most researched area by the Semantic Web reasoning community; *uncertainty in the Semantic Web*, where probabilistic reasoning is applied to the Semantic Web; and *vagueness in the Semantic Web*, where Semantic Web languages are extended with fuzzy logic concepts. At the end of this section, a summary with our most relevant conclusions about the related work is provided.

2.6.1. Deterministic Reasoning in the Semantic Web

Since Semantic Web languages like RDF and OWL are based on crisp logic, deterministic reasoning is the main area of research in Semantic Web reasoning. In this section are presented some of the best reasoners in this area.

Most of the reasoners in this section use *tableau-based* (also called *tableaux*) decision procedures for inference (Ralf Moller and Volker Haarslev 2008). These methods check the *satisfiability* (i.e., if there is an assignment of variables that make a statement evaluate to true) of an ontology by constructing a graph representation from the logical model of the ontology. This way, satisfiability can be checked by traversing the graph and checking for inconsistencies. This graph representation is

also useful to do more complex tasks, like concept satisfiability, classification, realization, and individual retrieval.

One of the most used reasoners is *Pellet*[6] (Sirin et al. 2007), an open source *Java* OWL DL Reasoner. It provides sound and complete reasoning with OWL DL, being capable of reasoning with assertional and terminological knowledge. It also provides reasoning with user defined datatypes, support for ontology debugging and integration with rules formalisms like *Semantic Web Rule Language*[7] (SWRL). It uses tableau Description Logic algorithms for consistency checking, making possible OWL DL concept satisfiability, classification, and realization.

FaCT++[8] (Tsarkov and Ian Horrocks 2006) is a Description Logic reasoner supporting OWL DL and OWL2. It is an open source reasoner made in C++. The main characteristics of *FaCT++* are the use of highly optimized tableau algorithms, making it a very efficient assertional and terminological knowledge reasoner. Reasoning in *FaCT++* is segmented in two tasks: preprocessing, when the ontology is loaded and some rewriting optimizations are made; and classification, where the taxonomy of named concepts is calculated using satisfiability checking techniques.

(Stocker and Smith 2008) proposed *Owlgres*[9], an open source scalable reasoner for OWL2 DL-Lite. It provides efficient reasoning and querying over scalable persistent data storages, like *relational database managements systems* (RDBMS) (e.g., *PostgreSQL*[10]). By combining Description Logic reasoning techniques with efficient data management and retrieval of RDBMS, *Owlgres* provides conjunctive query answering using a subset of *SPARQL*[11], *SPARQL-DL*, transforming *SPARQL* queries in *SQL* queries.

SHER[12] (Scalable High Expressive Reasoner) (Dolby et al. 2007) is an OWL DL reasoner developed by *IBM*[13]. It is optimized to high performance reasoning over millions of facts in OWL DL ontologies. It is mainly used to answer conjunctive and membership queries over ontological resources. Its high performance is obtained by

---

[6] http://clarkparsia.com/pellet/

[7] http://www.w3.org/Submission/SWRL/

[8] http://code.google.com/p/factplusplus/

[9] http://pellet.owldl.com/owlgres

[10] http://www.postgresql.org/

[11] http://www.w3.org/TR/rdf-sparql-query/

[12] http://domino.research.ibm.com/comm/research_projects.nsf/pages/iaa.index.html

[13] http://www.ibm.com/

summarizing ontological data into a very compact representation, and by refining this data by filtering unnecessary facts to the queries.

Hermit[14] (Boris Motik, Shearer, and Ian Horrocks 2007) is a *Java* OWL reasoner that uses a novel approach to Description Logic reasoning. This novel approach, called *hypertableau reasoning*, is a hybrid of resolution and tableau algorithms, being more efficient in some subsets of OWL ontologies.

KAON2[15] (Boris Motik and Ulrike Sattler 2006) is an OWL DL management system that provides efficient Description Logic reasoning. Unlike most of the Description Logic reasoners, *KAON2* does not use any tableau methods. Instead, the ontology is reduced to a *disjunctive Datalog program* and the inference is made in this representation. This way, some well know deductive database techniques like *magic sets* and *join-order optimization* can be applied in OWL DL reasoning, improving the results on answering some types of queries.

RACER[16] (Renamed ABox and Concept Expression Reasoner) (V. Haarslev and Möller 2003) is a Description Logic reasoner that can be used to reason over OWL DL ontologies. *RACER* uses optimized tableau calculus algorithms, supporting tasks like satisfiability, consistency, subsumption, and querying of ontological resources. The commercial version, *RacerPro*[17], also supports rules using SWRL and can be used as an HTTP reasoning server.

(Tsarkov et al. 2004) describe work on using *Vampire*[18] (A. Riazanov 2002), a general purpose first-order logic (FOL) reasoner, to reason with OWL DL. Since OWL DL is a subset of FOL, the authors translate the ontology to FOL and apply FOL reasoning techniques to it. They also translate SWRL rules into FOL and reason with those rules. The results were not as good as a general Description Logics reasoner, but since FOL is way more expressive than OWL DL, this approach can be useful in some more restrictive tasks, like testing and debugging of new tests.

---

[14] http://www.hermit-reasoner.com/

[15] http://kaon2.semanticweb.org/

[16] http://www.racer-systems.com/

[17] http://www.racer-systems.com/products/racerpro/index.phtml

[18] http://www.voronkov.com/vampire.cgi

2.6.2. Uncertainty in the Semantic Web

Some domains are uncertain by nature. Since Semantic Web languages, like OWL and RDF, are based on crisp logic, it is very difficult, if not impossible, to represent those domains in the Semantic Web. Most of the times, this uncertainty comes from our incapacity of asserting the veracity or falsity of a statement. This uncertainty is usually represented by a probability, i.e., a quantity representing our uncertainty.

There are currently two distinct approaches (Thomas Lukasiewicz and Umberto Straccia 2008) to add probabilistic knowledge in the Semantic Web: *Probabilistic Description Logics* tries to expand and modify the logic behind Description Logics with probabilistic knowledge; and *Probabilistic Semantic Web Languages* which tries to combine Semantic Web languages with probabilistic formalisms like Bayesian networks.

Probabilistic Description Logics

The most expressive Description Logic with probabilistic knowledge is $\mathcal{P}\text{-}SHOIN(D)$ (T. Lukasiewicz 2008) (Klinov and Bijan Parsia 2008). $\mathcal{P}\text{-}SHOIN(D)$ is a probabilistic extension to $SHOIN(D)$, the Description Logic behind OWL DL. Probabilities are represented by a new kind of axiom, called conditional constraints. *Conditional constraints* are composed by expressions of the form *(D|C)[l,u]*, where *D* is the evidence, *C* the conclusion, and *[l,u]* is a probability interval. There are two types of conditional constraints: *generic constraints*, representing probabilistic relations between classes; and *individual constraints*, representing probabilistic information about the belonging of individuals to certain classes. Currently, there are two reasoners that implement this Description Logic: *Pronto*[19] (Klinov 2008), a probabilistic OWL reasoner developed by the Pellet team, and *ContraBovemRufum*[20] (Nath and R. Moller 2008), a simple OWL probabilistic reasoner built on top of Racer. Recently, a tool[21] for the evaluation of probabilistic Description Logic reasoners was also developed. Since OWL Lite is a subset of OWL DL, its Description Logic, $SHIF(D)$, can also be extended with probabilistic knowledge, $\mathcal{P}\text{-}SHIF(D)$ (T. Lukasiewicz 2008).

---

[19] http://pellet.owldl.com/pronto

[20] http://www.sts.tu-harburg.de/~t.naeth/#Software

[21] http://www2.cs.man.ac.uk/~klinovp/projects/prevaldl/index.html

Very similar to the previous work is $\mathcal{P}\text{-}\mathcal{SHOQ(D)}$ (Giugno and Thomas Lukasiewicz 2002). This Description Logic is based on $\mathcal{SHOQ(D)}$, a Description Logic very similar to $\mathcal{SHOIN(D)}$. The only difference between them is that $\mathcal{SHOQ(D)}$ does not allow inverse properties.

There are also other Description Logics augmented with probabilistic knowledge, but none of those have the needed expressivity to comport the expressiveness of Semantic Web languages. An overview can be found on (Thomas Lukasiewicz and Umberto Straccia 2008).

Probabilistic Semantic Web Languages

Since the emergence of the first Semantic Web languages, some work has been done in representing probabilistic knowledge in those languages. Most of this work tries to add probabilistic capabilities to those languages without changing their logical foundations or their syntax, by combining them with known probabilistic formalisms.

In (Fukushige 2005) is proposed a simple vocabulary to represent probabilistic knowledge in RDF. This vocabulary is composed by a set of classes and properties representing elements of Bayesian networks, making possible to link RDF statements to those elements. This way, marginal and conditional probabilities about statements can be easily represented and reasoned using Bayesian networks. *pRDF* (Udrea, Subrahmanian, and Majkic 2006) is a formal probabilistic extension to a subset of RDF/S, allowing probabilistic knowledge about classes and properties of individuals. Unlike the previous work, *pRDF* implements its own probabilistic logic, making possible to reason over assertional knowledge in acyclic RDF graphs. (Holi and Hyvönen 2006) presented a framework for representing uncertainty in simple RDFS taxonomies. They were particular interested in computing the degrees of *subsumption*, i.e., overlap, between concepts. By attaching weights, called *masses*, to concepts, a Bayesian network is built. Using the Bayesian network prior and conditional probabilities, an overlap table between concepts is easily built by using *evidence propagation* algorithms.

PR-OWL[22] (Costa and Laskey 2005) is a probabilistic generalization of OWL based on *multi-entity Bayesian networks* (MEBNs) (Laskey and Costa 2005). MEBN logic combines Bayesian probability theory with first order logic, constructing Bayesian networks from parameterized fragments representing the probabilistic knowledge about a collection of related hypotheses. This way, probability distributions can be encoded in first-order theories. The main contribute of PR-OWL is the definition of

---

[22] http://www.pr-owl.org/

an upper ontology that guides the development of probabilistic ontologies in OWL using MEBNs.

(Ding, Peng, and Rong Pan 2006) proposed *BayesOWL*, a framework to represent and reason OWL uncertain knowledge using Bayesian networks. They provide a set of rules and procedures to translate OWL DL concept taxonomies (i.e., class axioms and logical relations between classes) into Bayesian networks. The main idea in this translation is to transform all the classes in variables and all the predicates in arcs between the respective classes. Special variables are inserted to facilitate the modeling of relations between concepts, and to avoid cycles. A simple approach to annotate OWL DL statements with conditional and marginal probabilities is also provided, as methods to automatically construct and refine conditional probability tables. The resulting Bayesian network preserves the semantics of the original ontology, and supports ontology reasoning both within and across ontologies. This framework was successfully applied in ontology mapping tasks (Rong Pan et al. 2005).

Similar to the previous work is *OntoBayes* (Yang and Calmet 2005), which combines OWL and Bayesian networks. They provide a simple method to annotate OWL with conditional, marginal, and full disjoint probabilities. Unlike the previous work, the Bayesian network is not automatically built. Users must annotate in OWL the dependencies between variables. This way, users are not restricted to a subset of OWL, like *BayesOWL*, but have the burden of annotate more elements. Given the OWL ontology, the probabilities and dependency annotations, a Bayesian network is easily built, and the inference is made on it. (Gu et al. 2004) also propose a similar approach to the last one, being more focused on reasoning over uncertain contexts represented in OWL.

(Henrik and Norbert 2006) describe work on probabilistic reasoning in two subsets of OWL Lite. They translate restricted OWL Lite ontologies into *Datalog*, a subset of first-order logic, and use a probabilistic extension of Datalog, *pDatalog*, to do probabilistic inference over the ontology. This approach was successfully used in automatic ontology matching tasks (Nottelmann and Umberto Straccia 2006). In (Predoiu and Stuckenschmidt 2007) a probabilistic framework for information integration and retrieval on the Semantic Web is proposed. Their approach uses *Bayesian Description Logic Programs*, a formalism that joins *Description Logic Programs* (DLP), a subset of *Datalog* without negation and without equality, with a fragment of *Bayesian Logic Programs*. In this representation, statements are translated to DLP rules with an attached probability. This way, a Bayesian network can be built from those annotated rules, providing a complete specification of the desired probability distribution.

2.6.3. Vagueness in the Semantic Web

Sometimes, real world domains are composed by imprecise or vague information. Again, Semantic Web languages are not ready to deal with this type of information.

For example, in those languages, how can we model the fact that some resource is "fast" or "tall"?

The problem with this situation is that we are dealing with *vague concepts*, that are more or less true, but we do not have a precise definition of them. For example, saying that *Ferrari is fast* depends on the velocity of the Ferrari, but we cannot say that this statement is completely true or false because we do not know which velocity is considered fast (i.e., fast is a vague concept). One way of representing vague concepts is by assigning a value to them, saying that this concept is true to some degree represented by that value. For example, by saying that *Ferrari is fast* with the degree of truth 0.8, we are saying that Ferrari is relatively fast but not completely fast.

One way of dealing with vagueness is by using *fuzzy set theory* and *fuzzy logic* (Klir and Yuan 1995). Instead of having a true/false value, statements with vague concepts have a truth value, usually between *[0, 1]*, where *0* and *1* represents binary false and true, respectively. In the last years, many approaches have been proposed to extend Description Logics with fuzzy set theory. Next, the most relevant approaches to this work are described. For a more complete overview see (Sanchez 2006) and (Thomas Lukasiewicz and Umberto Straccia 2008).

(Stoilos et al. 2005a) propose *f-OWL*, a fuzzy extension to OWL DL. In this extension, degrees of truthiness are added to OWL facts, representing the fuzziness of those facts. When a fact does not have any degree, it is interpreted as a binary true fact (i.e., degree of 1). In this approach, OWL syntax must be changed to cope with the addition of the membership degree. The reasoning is made by a new logic, called f-$\mathcal{SHOIN}$. This logic extends $\mathcal{SHOIN}$ (without datatypes) to deal with fuzzy set theory. (Stoilos et al. 2005b) also proposed f-$\mathcal{SHIN}$, a simpler fuzzy description logic that is base of *FiRE*[23], a fuzzy reasoner for the Semantic Web. A more complete fuzzy extension of $\mathcal{SHOIN}$ was proposed by (U. Straccia 2006a). Although the principles are very similar, this approach subsumes the previous one, being capable of supporting all the OWL DL language.

Another fuzzy Description Logic reasoner is *fuzzyDL*[24] (Bobillo and U. Straccia 2008), a fuzzy extension to $\mathcal{SHIF(D)}$, the Description Logic behind OWL Lite. This reasoner supports various membership functions and distinct fuzzy logics, like *Zadeh semantics* and *Lukasiewicz logic*. This reasoner also supports classical Description Logic reasoning, being not restricted to fuzzy reasoning. In (U. Straccia 2006b), a

---

[23] http://www.image.ece.ntua.gr/~nsimou/FiRE/

[24] http://gaia.isti.cnr.it/~straccia/software/fuzzyDL/fuzzyDL.html

fuzzy extension to a restricted subset of OWL DL, called *DL-Lite*, is proposed. This extension, called *f-DL-Lite*, supports fuzzy queries over fuzzy knowledge bases. (J. Z. Pan et al. 2007) extended the previous work, proposing new query answer languages to query fuzzy knowledge bases, by extending SPARQL to support membership degrees. The work is implemented in *ONTOSEARCH2*[25], a search and query engine for the Semantic Web.

2.6.4. Conclusions

Through the analysis of the previous related work, some general conclusions can be made:

- *There is little work on dealing with uncertainty and vagueness in the Semantic Web*. Even if these two areas are of the most importance to the future of the Semantic Web, only recently the scientific community has started to research them (the first publications about the subjects are from 2005). In the same year, a dedicated workshop about those subjects (*International Workshop on Uncertainty Reasoning for the Semantic Web*[26]) was created;

- *Deterministic reasoning is more computationally efficient than all the other approaches*. Some of the systems seen in Section 2.6.1 support efficient reasoning over millions of ontological facts, while other systems, like *Pronto*, are restricted to only hundreds of them. The main reason is that deterministic reasoning is the main area of research in Semantic Web reasoning, having already developed very efficient algorithms, mainly those based on tableau procedures. These algorithms were already being improved in other older fields, like Description Logics;

- *Probabilistic Semantic Web languages are, usually, more computationally efficient than the other uncertain and vague approaches*. The main reason is that most of them use formalisms like probabilistic graphical models, which are well known and provide efficient reasoning mechanisms. Probabilistic and fuzzy Description Logics usually lack of efficient reasoning mechanisms, and their applications to real world domains are also not well studied;

[25] http://www.ontosearch.org/

[26] http://c4i.gmu.edu/ursw/

- *All the works on uncertainty and vagueness in the Semantic Web rely on the principle that the uncertainty or vagueness of the ontology is already asserted.* To our knowledge, there is no work on extracting automatically this information from the ontology, or from other knowledge representations. However, there are many ontologies that are uncertain or vague by nature, but do not have any type of information denoting that fact. This fact leads to the need of develop efficient mechanisms to learn this information.
- *All the works on Probabilistic Semantic Web languages rely on probabilistic formalisms, like Bayesian networks, which do not allow cycles.* However, in many domains, knowledge is cyclic (e.g., the relations *FatherOf* and *SonOf* are cyclic). This fact limits the usability and expressiveness of these approaches in real world domains.

These conclusions identify the main problems and debilities of the related work. This information will be used in the definition of our proposed approach (Chapter 3).

3.   Learning and Reasoning about Uncertainty in the Semantic Web

The objective of this thesis is to study mechanisms to perform probabilistic reasoning in the Semantic Web (T. Berners-Lee, J. Hendler, and Lassila 2001). For this purpose, we use Markov logic (Pedro Domingos, Stanley Kok, et al. 2008), a novel representation formalism that combines logic and probabilistic graphical models, and ontologies represented in OWL2 (Grau et al. 2008), the Web Ontology language proposed by the W3C. Before explaining our approach, we first need to clarify why we are using Markov logic and OWL2.

Why Markov Logic

As noted in Section 2.4.6, the most relevant Semantic Web languages to describe ontologies are based on Description Logics (Baader et al. 2007). These Description Logics follow a model-theoretic semantics, and therefore can be usually interpreted as a set of formulas in first-order logic. To perform probabilistic reasoning over these languages, we need formalisms that allow first-order logic and probabilities. These formalisms must be also prepared to cope with domains with a high relational complexity, as the ones presented in the Semantic Web vision (T. Berners-Lee, J. Hendler, and Lassila 2001). These requirements lead us to the field of statistical relation learning (Lise Getoor and Ben Taskar 2007). In this field, there are two main approaches that combine the full power of first-order logic and probabilistic graphical models: Markov logic and Bayesian logic (BLOG) (Milch et al. 2007). Between those two, Markov logic was chosen for a set of reasons:

- BLOG models are based on Bayesian networks, and therefore do not allow cycles. Since the Semantic Web domain is characterized by cyclic relations between entities (e.g., the relations *FatherOf* and *SonOf* are cyclic), this fact can restrict the use of BLOG in some ontologies. It is studied (Ding, Peng, and Rong Pan 2006) that these cycles can be removed by the introduction of auxiliary variables. However, this is a difficult task that increases the model complexity, and can be only used on a small subset of some Semantic Web languages;

- Even if BLOG allows first-order knowledge, models are procedurally defined by programs, an unnatural way of defining first-order knowledge. To model a simple fact, complex constructors like *guaranteed object statements* and *dependency statements* must be declared, while in Markov logic it suffices to declare the first-order logic formulas of the domain, and their weights;

- Several algorithms for learning and reasoning in Markov logic were studied (e.g., (P. Singla and P. Domingos 2005) (S. Kok and P. Domingos 2005) (P. Singla and P. Domingos 2006a) (H. Poon and P. Domingos 2006)). In BLOG only reasoning was deeply studied (Milch and S. Russell 2006) (Milch et al.

2008), and there is no work on learning the parameters or the structure of BLOG models.

Why OWL2

In Section 2.4.6, we presented four main Web Ontology language versions: OWL Lite, OWL DL, OWL Full, and OWL2. The last one was chosen for a set of reasons:

- Just like OWL Lite and OWL DL, OWL2 is decidable. This fact is determinant in the search of efficient reasoning mechanisms;
- OWL2 is a very expressive language, which subsumes OWL Lite and OWL DL;
- OWL2 provides improved annotation mechanisms, making the annotation of axioms easier. Annotations are relevant in our domain, since it is the preferred way to attach uncertainty information to axioms;
- Some of the most used Semantic Web tools, like Protégé[27] and OWLAPI[28], already support OWL2 and are encouraging the Semantic Web community to use this new language.

From OWL to Markov Logic

The first step in use Markov logic capabilities to reason about uncertainty in the Semantic Web is to transform Semantic Web's representation languages, in our case OWL2, into Markov Logic Networks (MLNs). As seen in Section 2.5.1, a MLN is composed by a set of weighted first-order logic formulas. So, we must define where these formulas and weights come from.

Formulas

OWL2 is based on the Description Logic $\mathcal{SROIQ(D)}$ (Grau et al. 2008). One characteristic of Description Logic languages is that they follow a *model-theoretic* semantics (Baader et al. 2007), and therefore can (in most of the cases) be interpreted as formulas in first-order logic. The main idea behind this interpretation is that concepts correspond to unary predicates, roles to binary predicates, and individuals correspond to constants. In our case, $\mathcal{SROIQ(D)}$ can be easily interpreted as first order formulas. Table 9 provides some of these interpretations (see Appendix I for the full interpretation).

---

[27] http://protege.stanford.edu/

[28] http://owlapi.sourceforge.net/

OWL2 Axiom	First-order logic formula
$SubClassOf(CE_1, CE_2)$	$\forall x : CE_1(x) \Rightarrow CE_2(x)$
$TransitiveProperty(OPE)$	$\forall x, y, z : OPE(x, y) \wedge OPE(y, z) \Rightarrow OPE(x, z)$
$ClassAssertion(CE, a)$	$CE(a)$

Table 9. Examples of first-order logic interpretations of OWL2 axioms.

Weights

In the next sections, we explore several sources for acquiring weights (Figure 12). First, we explore the cases when the ontologies are already annotated with some kind of uncertainty values that can be interpreted as weights. If those values are already weights, the interpretation is straightforward, and no posterior processing is needed (Section 3.1). However, in the cases where those values are probabilities, we must transform them in weights (Section 3.1.1).

Second, we explore the cases where ontologies do not have any type of uncertainty annotation available. If the ontology contains individuals, we can use those individuals to learn the weights using the weight learning capabilities of Markov logic (Section 3.2). In the cases where the ontologies do not have individuals, resources like textual corpus and web search engines can be used to learn individuals or derive automatically the probability of axioms (Section 3.3).
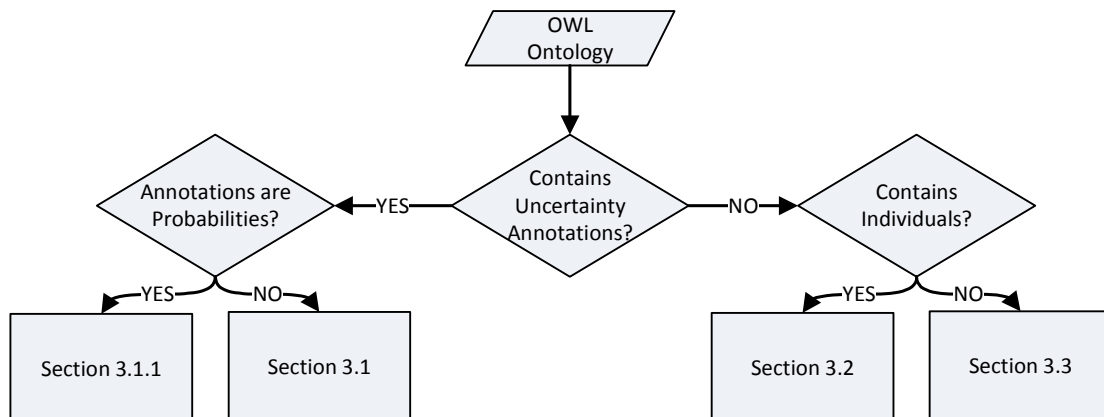
Figure 12. Explored ontology weight sources**.**

3.1. Probabilistic Reasoning in Uncertainty-annotated Ontologies

Ontology axioms can be annotated with a value representing its uncertainty (usually a weight or a probability) (e.g., a certain class has *X%* probability of being subclass of another). This allows ontology engineers to build uncertain ontologies with their own knowledge about the domain. However, this kind of reasoning is not only interesting in this case. There are already domains with ontologies with some kind of uncertainty associated:

- In the field of *ontology learning* from text corpus (Maedche 2002), the resulting ontologies usually have a probability associated with the confidence on the asserted relation (e.g., based on the corpus, there is *X%* probability that two concepts are related);

- In *ontology mapping, alignment, and matching* tasks (Euzenat and Shvaiko 2007), most of the relations found between distinct ontologies are probabilistic (e.g., two concepts in two different ontologies are the same with *X%* probability);

- Ontologies are being used to express *user context* when using a certain application (e.g., (Kersten and Murphy 2006)). For example, an application that recommends actions to the user can use ontologies to assert what are the most interesting concepts to the current context. This is usually done by assigning weights or probabilities to the ontology concepts and relations (e.g., in a word processor, when the user is writing about dogs, the concept *Dog* is *X%* more important than the concept *Cat*).

Example

Suppose we have a simple hand-made ontology (Table 10) that models a domain about birds and their capability to fly, composed by 3 classes (*Bird*, *FlyingAnimal*, and *Penguin*) and 2 individuals (*Tim* and *Tweety*). Given the translation to first-order logic in Table 11, using Markov logic we can query for the conditional probabilities of the ontology individuals fly:

$$P\big(FlyingAnimal(Tim)\big) = 0.87$$
$$P\big(FlyingAnimal(Tweety)\big) = 0$$

Axiom	Weight
$SubClassOf(Bird, FlyingAnimal)$	1.8
$SubClassOf(Penguin, ComplementOf(FlyingAnimal))$	10
$SubClassOf(Penguin, Bird)$	10
$ClassAssertion(Tim, Bird)$	
$ClassAssertion(Tweety, Penguin)$	

Table 10. Flying Animals ontology.

First-order logic formulas	Weight
$\forall x : Bird(x) \Rightarrow FlyingAnimal(x)$	1.8
$\forall x : Penguin(x) \Rightarrow \neg FlyingAnimal(x)$	10
$\forall x : Penguin(x) \Rightarrow Bird(x)$	10
$Bird(Tim)$	
$Penguin(Tweety)$	

Table 11. Flying Animals ontology in first-order logic.

3.1.1. Probabilities instead of Weights

In most of the previously referred cases, the uncertainty is represented as a probability. In Markov logic, weights have a direct correspondence with probabilities if they are interpreted as log odds:

$$w_i = log\frac{p_i}{1 - p_i},$$

where p_i is the probability of the formula F_i, and w_i its corresponding weight.

However, if F_i shares variables with other formulas, as typically is the case, this correspondence cease to hold, since the weight of F_i is influenced not only by its probability, but also by the other formulas that share the same variable. In this case, the probabilities of all formulas collectively determine all the weights. One solution to this problem is to treat formulas' probabilities as *empirical frequencies* and learn their weights using the algorithms of Section 2.5.3. For example, if we have the formula $\forall x : Bird(x) \Rightarrow FlyingAnimal(x)$ and we know that it is true in 99% of the cases, we create 99 individuals that are both *Birds* and *FlyingAnimals*, and 1 that is a *Bird* but not a *FlyingAnimal*, and then learn its weight with those individuals. However, this solution is unfeasible in large and complex domains for a set of reasons:

- We have to create many individuals, especially if the domain has many types of individuals (we have to have distinct individuals for each one of the types) and/or we want to have a good approximation of the desired probability (e.g., in the previous example, if we had the probability 99.1%, we needed 1000 *Birds*, 991 of them *FlyingAnimals*). The more individuals we have, more difficult is the weight learning;
- We can have very difficult and complex formulas, making the translation to empirical frequencies very difficult to achieve (e.g., $\exists x \forall y : A(x) \wedge (B(x) \wedge \neg C(x)) \vee P(x,y) \Rightarrow x \neq y$);
- It can be impossible to create a proper empirical frequency in cases when formulas contradict other formulas. For example, if we have $\forall x : A(x) \wedge B(x) \Rightarrow C(x)$ and $\forall x : A(x) \Rightarrow \neg C(x)$ both with 100% probability, we cannot create a correct empirical frequency, because an individual of class A cannot be at the same time both C and $\neg C$. These types of formulas could arise in tasks like *entity matching*, where there could be contradictions between formulas and individuals.

A better solution can be achieved by analyzing one of the weight learning algorithms used in Markov logic. The *discriminative weight learning* algorithm (P. Singla and P. Domingos 2005) maximizes the conditional likelihood of some query predicates taking only in account the evidence atoms *X* and query atoms *Y*.

The gradient of the conditional likelihood with respect to the weights is defined by

$$\frac{\partial}{\partial w_i} \log P_w(y|x) = n_i(x, y) - E_w[n_i(x, y)],$$ **3.2**

where $n_i(x, y)$ is the number of true groundings of clause i in data, and $E_w[n_i(x, y)]$ is the expected number of true groundings of clause i according to the model, value that is approximated by the counts of the most probable state of *y* given *x*. The number of true groundings is acquired by counting the number of groundings of the clause that are true in the data in respect to the model. However, if we have the probability of the clause[29], this value can be easily calculated as

$$n_i(x, y) = count(i) * p_i,$$ **3.3**

where $count(i)$ is the total number of groundings of clause i, and p_i the probability of the clause. This way, instead of relying on the individuals to acquire the number of true groundings of the clause, we automatically calculate that value with the desired probability.

This solution is more feasible than the previous one, since it needs fewer individuals (we only need one individual for each individual type) and can be applied in any type of first-order logic formula.

Correctness Study

In this section, we study the behavior of the proposed approach in comparison with the empirical frequency approach. The main objective is to check if the proposed solution derives the same results as the training by empirical frequencies. For this purpose, we created four example MLNs (Table 12), with 10 individuals each, where the formulas where annotated with probabilities. Next, we tested all the possible combinations of probabilities for those formulas, with increases of 0.1, and compared the weights generated by both approaches. As we can see in Table 13, both solutions provide very similar weights, verifying the correctness of the proposed approach.

---

[29] In Markov logic, formulas are usually transformed in clausal form, generating a set of clauses. In this case, the probability of the formula is equally divided by its clauses.

MLN	First-order logic formulas
1	$\forall x : A(x) \Rightarrow B(x)$
2	$\forall x : A(x) \wedge B(x) \Rightarrow C(x)$
3	$\forall x : A(x) \Rightarrow B(x)$ $\forall x : A(x) \Rightarrow C(x)$
4	$\forall x : A(x) \wedge B(x) \Rightarrow C(x)$ $\forall x : D(x) \Rightarrow \neg C(x)$ $\forall x : B(x) \Rightarrow D(x)$

Table 12. Correctness study MLNs.

MLN	Mean Absolute Weight Difference
1	0.0575
2	0.025
3	0.0625
4	0.05

Table 13. Correctness study results. Mean absolute weight difference is the arithmetic mean of the absolute difference between the weights generated by both solutions with the MLNs of Table 12. A small value indicates that both solutions are similar.

3.1.2. Experimentation

In this section, we present two probabilistic domains that can exploit the previously defined capabilities. First, we analyze a domain about gesture-based affective information recognition, where the probabilities where asserted by field experts. Next, we explore the applicability of Markov logic to reason with automatic learned ontologies.

The Body Gesture Experiment

One of the most interesting tasks in *Affective Computing* (Picard 1997) is to predict the affective state of a person based on its gestures. (Rehman Abbasi, V. Afzulpurkar, and Uno 2008) recorded the unintentional movements of students during a lecture, and manually labeled the movements in 7 gestures (*Head Scratch*, *Nose Itch*, *Lip Touch*, *Eye Rub*, *Chin Rest*, *Lip Zip*, and *Ear Scratch*), corresponding to 6 distinct affective states (*Recalling*, *Satisfied*, *Thinking*, *Tired*, *Bored*, and *Concentrating*). Based on their results, we developed a simple probabilistic ontology (Table 14).

Axiom	Probability
$SubClassOf(HeadScratch, Recalling)$	1
$SubClassOf(ChinRest, Thinking)$	0.9
$SubClassOf(EyeRub, Tired)$	0.81
$SubClassOf(LipTouch, Thinking)$	0.8875
$SubClassOf(NoseItch, Satisfied)$	0.775
$SubClassOf(LipZip, Bored)$	1
$SubClassOf(EarScratch, Concentrating)$	0.8333

Table 14. Affective state prediction probabilistic ontology.

Next, we defined several sets of individuals and used MC-SAT to perform nine probabilistic queries (Table 15). Some remarks about the results:

- In the first six results, we can see that the probabilities are similar to those expressed in the ontology. The residual differences are derived by the fact that both learning and reasoning are made using approximate algorithms;

- Result number 2 gives a probability 0.07 less than the desired probability of 0.9. This is not only derived by the use of approximate algorithms, but also by the fact that there is other axiom that leads to the conclusion *Thinking* (axiom number 4). Since we do not know if this individual also belongs to the class *LipTouch*, there is a small probability that axiom 4 is also true, and since that axiom contains a smaller weight, the final probability decreases. The inverse occurs in result number 4, where the probability increases in comparison to the expected probability.

- As expected, the probability of result number 8 is greater than the ones with its assertions alone (results 2 and 4).

Set Number	Assertions	Queries and Results
1	$ClassAssertion(A, HeadScratch)$	$Recalling(A) = 0.96$
2	$ClassAssertion(A, ChinRest)$	$Thinking(A) = 0.83$
3	$ClassAssertion(A, EyeRub)$	$Tired(A) = 0.81$
4	$ClassAssertion(A, LipTouch)$	$Thinking(A) = 0.8$
5	$ClassAssertion(A, LipZip)$	$Bored(A) = 0.94$
6	$ClassAssertion(A, EarScratch)$	$Concentrating(A) = 0.79$
7	$ClassAssertion(A, EarScratch)$ $ClassAssertion(A, ChinRest)$	$Thinking(A) = 0.84$ $Concentrating(A) = 0.83$
8	$ClassAssertion(A, LipTouch)$ $ClassAssertion(A, ChinRest)$	$Thinking(A) = 0.89$

Table 15. Affective state prediction results.

The Ontology Learning Experiment

As previously referred, one of the fields that already produces probabilistic ontologies is the field of ontology learning. In this experiment, we reason about taxonomies automatically learned from web search engines.

Using the *lexico-syntactic* patterns defined by (Hearst 1992) (for more information about those patterns, see Section 3.3.1), we developed a simple system that receives the root of the taxonomy and uses a web search engine to infer its descendants until a pre-defined depth. The subsumption relations receive a confidence value using the following metric (McDowell and Cafarella 2008):

$$Score(i,c) = \frac{count(i,c)}{count(i)}. \qquad \textbf{3.4}$$

Here, $count(i,c)$ is the number of times that class i appears subsumed by c, and $count(i)$ is the total number of times that class i appears subsumed by any class. This metric gives a value in the interval [0,1] and can be roughly interpreted as a probability (i.e., the probability of choosing the class c as the subsumer of i). Using the Yahoo Boss API (see Table 34 for more information about this API), we created a taxonomy with the class *Animal* as root, with a tree-depth of 3 levels. A graphical representation of an excerpt of this taxonomy, with the respective probabilities, is seen on Figure 13. This taxonomy is easily represented in OWL2 using the $SubClassOf$ relation.
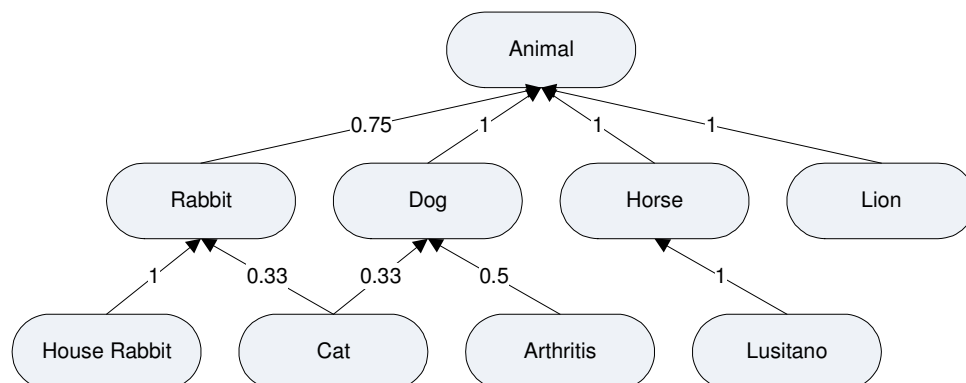
Figure 13. Automatically learned probabilistic taxonomy about animals. Directed edges represent the *SubClassOf* relation (e.g., *SubClassOf(Lusitano,Horse)*).

Using this taxonomy and some sets of example individuals, we can use MC-SAT to query for the conditional probabilities of some classes. The most interesting results are in the next table.

Set Number	Assertions	Queries and Results
1	$ClassAssertion(A, Lusitano)$	$Horse(A) = 0.9$ $Animal(A) = 0.87$ $HouseRabbit(A) = 0.17$
2	$ClassAssertion(A, Cat)$	$Dog(A) = 0.24$ $Rabbit(A) = 0.29$ $Animal(A) = 0.76$
3	$ClassAssertion(A, HouseRabbit)$	$Rabbit(A) = 0.81$ $Animal(A) = 0.58$

Table 16. Most interesting reasoning results of Figure 13 taxonomy.

Using the same approach, we can reason about other automatically extracted taxonomies (Figure 14, Figure 15, and Figure 16). Some example queries and their results can be seen on Table 17.
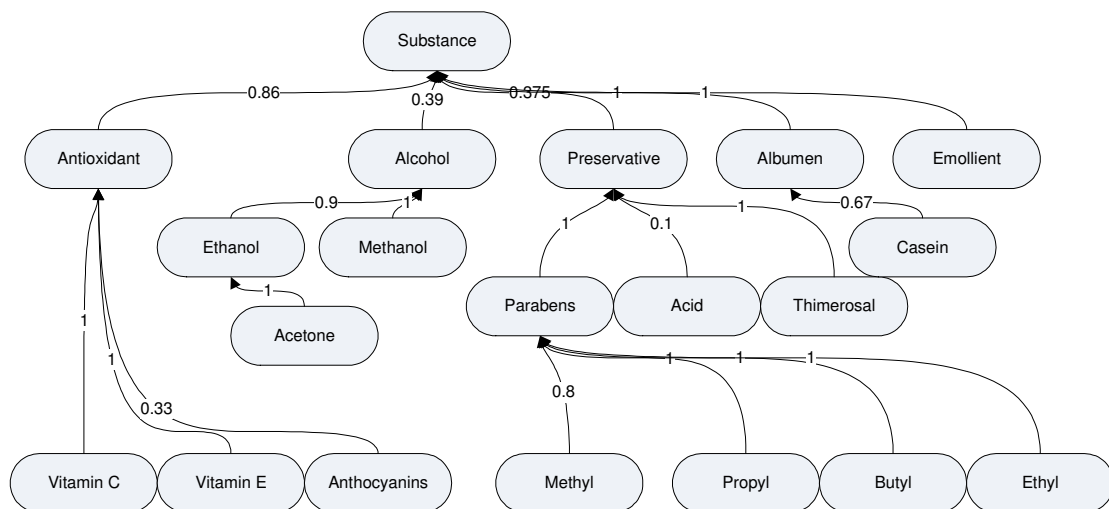
Figure 14. Automatically learned probabilistic taxonomy about substances. Directed edges represent the *SubClassOf* relation.
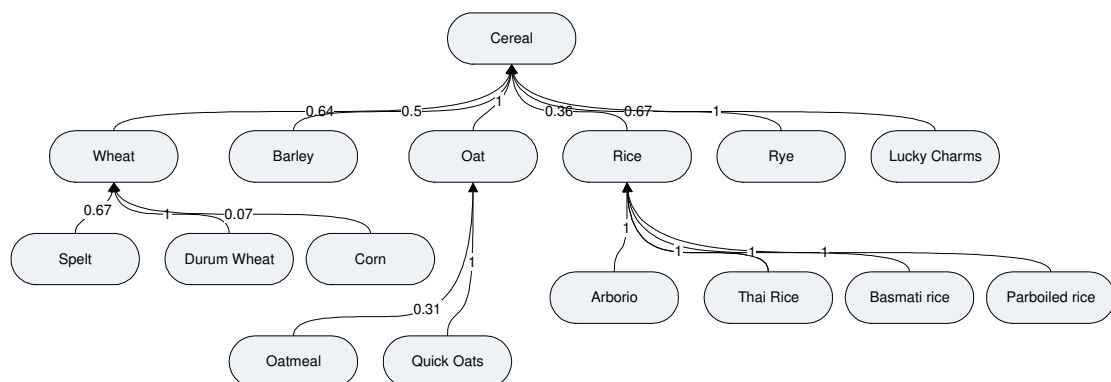
Figure 15. Automatically learned probabilistic taxonomy about cereals. Directed edges represent the *SubClassOf* relation.
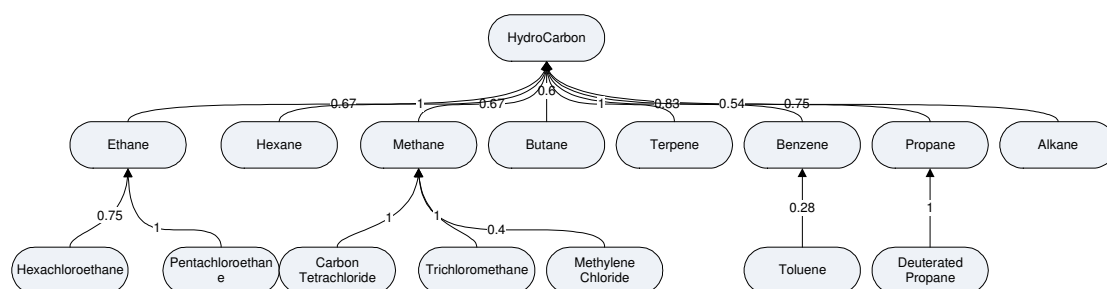
Figure 16. Automatically learned probabilistic taxonomy about hydrocarbons. Directed edges represent the *SubClassOf* relation.

Set Number	Assertions	Queries and Results
1	$ClassAssertion(A, Acetone)$	$Ethanol(A) = 0.91$ $Alcohol(A) = 0.81$ $Substance(A) = 0.58$
2	$ClassAssertion(A, OatMeal)$ $ClassAssertion(B, BasmatiRice)$ $ClassAssertion(C, Spelt)$	$Oat(A) = 0.11$ $Cereal(A) = 0.24$ $Rice(B) = 0.95$ $Cereal(B) = 0.33$ $Wheat(C) = 0.72$ $Cereal(C) = 0.1$
3	$ClassAssertion(A, Pentachloroethane)$ $ClassAssertion(B, Hexane)$	$Ethane(A) = 0.9$ $HydroCarbon(A) = 0.48$ $HydroCarbon(B) = 0.86$

Table 17. Most interesting reasoning results of Figure 13, Figure 14, and Figure 15 taxonomies.

3.2. Probabilistic Reasoning using Ontology Individuals

In the last section, we adopted the principle that ontologies were somehow annotated with some kind of uncertainty information. However, these situations only occur in restricted domains, mainly those where these ontologies were built automatically by machines. In fact, it is studied (Tversky and Kahneman 1974) that humans are not good at either producing or perceiving concepts related to uncertainty, like probabilities. And even if humans were good at perceiving these types of concepts, creating and maintaining large uncertainty annotated ontologies can be a cumbersome and difficult task, invalidating all the gains that could arise from the annotation. These facts raise the importance of developing mechanisms to learn this uncertainty automatically. This can be useful not only to help users when creating uncertain ontologies, but also to gain access to the vast number of non-uncertainty annotated ontologies already available.

As noted in Section 2.5.3, in Markov logic, formulas' weights can be learned generatively or discriminatively through example data. This example data usually is composed by individuals of the domain and their relations. In OWL2, individuals

correspond to the ABox of the ontology, and therefore they can be used to learn the formulas weights by interpreting them as ground atoms.

Example

Assume a simple ontology about birds (Table 18), with several example birds. Using Markov logic's generative learning, we learned the weights present in Table 19. With that information, using MC-SAT, some probabilistic queries can be made (Table 20).

Axiom
$SubClassOf(Bird, FlyingAnimal)$
$SubClassOf(Penguin, ComplementOf(FlyingAnimal))$
$SubClassOf(Penguin, Bird)$
$ClassAssertion(Tim, Bird)$
$ClassAssertion(Tom, Bird)$
$ClassAssertion(Tweety, Bird)$
$ClassAssertion(Tweety, Penguin)$
$ClassAssertion(Tim, FlyingAnimal)$
$ClassAssertion(Tom, FlyingAnimal)$

Table 18. Flying Animals ontology, with several example birds.

Axiom	Weight
$SubClassOf(Bird, FlyingAnimal)$	0.88
$SubClassOf(Penguin, ComplementOf(FlyingAnimal))$	1.45
$SubClassOf(Penguin, Bird)$	0

Table 19. Learned weights for Table 18 ontology.

Assertion	Query and Result
$ClassAssertion(A, Penguin)$	$FlyingAnimal(A) = 0.17$
$ClassAssertion(B, Bird)$	$FlyingAnimal(B) = 0.7$

Table 20. Most interesting reasoning results of Table 18 ontology.

3.2.1. Experimentation

In this section, we explore several domains where individuals can be used to learn the uncertainty of the ontology: a financial ontology about a bank and its operations, a web based social network, and two machine learning datasets transformed into ontologies.

The Financial Experiment

Evaluation Procedure and Data Set. Uncertainty reasoning is very important in discovering hidden knowledge in *risk assessment* domains. In this experiment, we use a financial ontology, GoldDLP[30], to assess the risk of certain financial operations. In this ontology, there is information about a bank that offers services like loans and credit cards to private persons. The ontology contains 116 class and property axioms and 297 individuals, mainly distributed between accounts, clients, credit cards, and loans. One of the most interesting tasks in this domain is to determine if a given loan is a problematic loan. There is an OWL class responsible for that information, named *ProblemLoan*, and some axioms about that class (e.g., *ProblemLoan* is the complement of *OkLoan*). The main task in this experiment is to determine each loan's probability of being a *ProblemLoan*.

Experimental Results. Using generative learning and MC-SAT, we found that nine loans have a probability >90% of being a *ProblemLoan*. If we compare the results (Table 21) with a non-probabilistic reasoner, like *Pellet*[31] (Sirin et al. 2007), these are the same nine individuals identified deterministically by it. However, our approach returns some more interesting results that were not identified by Pellet. All the other loans have a probability between 45-48% of being a *ProblemLoan*. This information is valuable because, roughly speaking, it demonstrates that any loan has an associated probability of being a problematic loan. This kind of results cannot be achieved using non-probabilistic reasoning, and therefore demonstrates the necessity of probabilistic reasoning to have a more profound understanding about the domain. However, if we use an existent Semantic Web probabilistic reasoner (e.g., *Pronto*[32] (Klinov 2008)), its results are the same of a non-probabilistic one, since the ontology does not contain any information about the uncertainty of its axioms.

---

[30] http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm

[31] http://pellet.owldl.com/

[32] http://pellet.owldl.com/pronto

Individual	Probability	Pellet
loan5148	0.48	False
loan5363	0.93	True
loan5549	0.48	False
loan5582	0.45	False
loan5868	0.97	True
loan6007	0.46	False
loan6202	0.98	True
loan6227	0.95	True
loan6297	0.97	True
loan6585	0.45	False
loan6599	0.95	True
loan6995	0.97	True
loan7130	0.97	True
loan7137	0.97	True
loan7154	0.47	False
loan7171	0.47	False

Table 21. Financial experiment results comparison between the proposed approach (*Property* column) and a deterministic reasoner (*Pellet).*

The Social Network Experiment

One of the most used Semantic Web vocabularies is the *Friend of a Friend*[33] (FOAF) vocabulary. This vocabulary allows describing social network data (i.e., persons and their relations) in OWL, with special incentive in linking users from different social networks. There are several web-based social networks that provide information about their users in FOAF (see *Mindswap*[34] for a comprehensive list), and some projects are already exploiting that information (e.g., Google Social Graph API[35]).

The objective of this experiment is to use Markov logic to explore the relational structure of FOAF networks. As data set, we choose *Advogato*[36], a social network of free software developers. Advogato provides three interesting FOAF properties for our analysis: *foaf:knows(x,y)*, meaning that user x knows user y; *foaf:currentProject(x,y)*, meaning that user x is currently working in project y; and *foaf:member(x,y)*, meaning that user x is member of the group y. After gathering and

---

[33] http://www.foaf-project.org/

[34] http://trust.mindswap.org

[35] http://code.google.com/apis/socialgraph/

[36] http://advogato.org/

processing all the available FOAF profiles, we had a total of 6688 individuals, representing 4198 users, 2487 projects, and 3 groups. Based on the *Link Mining* literature (Lise Getoor and Diehl 2005)(Lise Getoor 2003), we identified three interesting tasks to our experiment: link prediction, link-based classification, and link-based cluster analysis.

Link Prediction

Link prediction (Lise Getoor and Diehl 2005) is the problem of predicting the existence of a link between two objects based on the relations of the object with other objects. In our domain, we are particularly interested in predicting the acquaintance between users, i.e., the *foaf:knows* property. For this purpose, based on our common sense about the domain, we defined three simple rules to perform this task:

Weight	Formula
0.09	$\forall x, y, z : knows(x, y) \wedge knows(y, z) \Rightarrow knows(x, z)$
2.70	$\forall x, y : knows(x, y) \Leftrightarrow knows(y, x)$
1.11	$\forall x, y, z : currentProject(x, z) \wedge currentProject(y, z) \Rightarrow knows(x, y)$

Table 22. Link prediction rules.

The first two rules define *knows* as a *transitive* and *symmetric* property, respectively, while the last rule states that if two persons work on the same project, they probably know each other. Weights were learned generatively with all the individuals available. To better describe the results of the link prediction, we developed a simple artificial example composed by 9 users and 3 projects (Figure 17). Next, using MC-SAT, we queried for the conditional probabilities of the *foaf:knows* property for all those users. Results can be seen on Table 23.
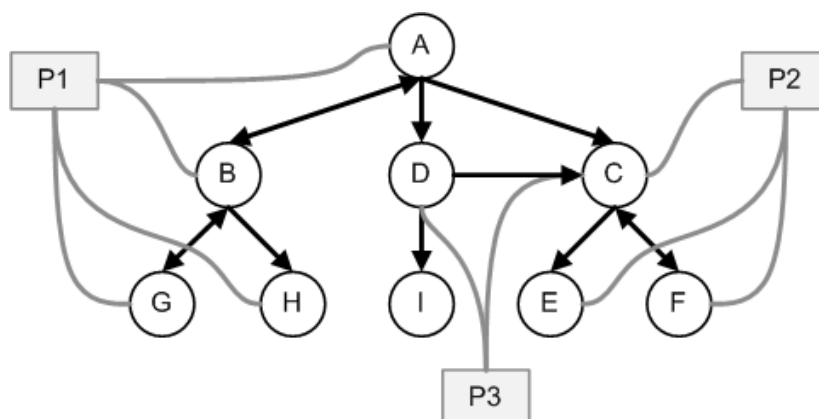
Figure 17. Graphical representation of the artificial example. Users are represented by circles (A-I) and projects by squares (P1-P3). Black directed edges represent the *foaf:knows* relation, while gray undirected edges represent the *foaf:currentProject* relation.

Query	Result
knows(A,G)	0.90
knows(A,F)	0.54
knows(D,A)	0.94
knows(C,A)	0.92
knows(C,D)	0.98
knows(H,F)	0.47

Table 23. Most relevant results of the previous example.

Some interesting results can be seen in this example:
- *knows(A,G)* is greater than *knows(A,F)*, even if both users are at the same distance from *A*. The only difference between them is that *G* works in the same project than *A*, getting a bigger probability;
- *knows(D,A)*, *knows(C,A)*, and *knows(C,D)* have big probabilities, mostly because the symmetry of *knows*. However, the probability of *knows(C,D)* is the greatest, since both users also work in the same project, *P3*;
- Since *H* and *F* does not share any direct connection, the probability of *knows(H,F)* is low, but not null.

Link-based Classification

The main task in *link-based classification* (Lise Getoor 2003) is to predict the category of an object based on the relations of that object with other objects. In our domain, there are three groups of users related to the experience of the user in the community: *Apprentice*, *Journeyer*, and *Master*. These groups are expressed through the *foaf:member* property. The objective of this experiment is to predict each user's group based on their connections to other users. For this purpose, we defined another simple rule that uses the relationship between users expressed on the three rules of Table 22:

Weight	Formula
0.19	$\forall x, y, z : knows(x,y) \wedge member(x,z) \Rightarrow member(y,z)$

Table 24. Link-based classification rule.

This rule states that the group of a user is influenced by the groups of the users that he knows. The weight of the rule was learned generatively in conjunction with the three rules of the previous experiment (their weights remained very similar). Next, we extracted a sub-network composed by 172 users (11 Apprentices, 55 Journeyers, 93 Masters) and 54 projects and randomly removed the group information to 27% of the users (i.e., 47 users). With the rules of Table 22 and Table 24 and the sub-network individuals, we used MC-SAT to predict the membership of the missing group users. The results, using metrics 3.5, 3.6, 3.7, and 3.8, can be seen in the next table.

$$Specificity = \frac{True\ Negatives}{True\ Negatives + False\ Positives} \qquad \textbf{3.5}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \qquad \textbf{3.6}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \qquad \textbf{3.7}$$

$$F - Measure = \frac{2 * (Precision * Recall)}{Precision + Recall} \qquad \textbf{3.8}$$

Group	Specificity	Precision	Recall	F-measure
Apprentice (4)	0.98	0	0	0
Journeyer (15)	0.97	0.83	0.33	0.48
Master (28)	0.37	0.7	1	0.82
Weighted Avg	0.61	0.68	0.70	0.64

Table 25. Link-based classification results. Between brackets is the number of individuals of the group.

Good results can be achieved on predicting user's groups taking only in account the relational structure of the network. The bad results on predicting the *Apprentice* group are probably derived from the small number of elements of that group in the test network. The results could be probably improved if other non-relational information about users was provided (e.g., nationality, age, sex).

Link-based cluster analysis

In the last experiment, we had seen how to classify users in a set of predefined groups. However, in some cases, the information about groups is not available and we still need to segment the users. The goal of *link-based cluster analysis* (Lise Getoor 2003) is to cluster objects into groups that show similar relational characteristics. In our domain, it is interesting to cluster users given their acquaintances with other users. For this task, we can use the three rules presented in the link prediction task (Table 22), since they can gave us a relational matrix of the *foaf:knows* property for all the users (i.e., the probability of all the users know each other). Using the same sub-network of the last task (172 users and 54 projects), we used MC-SAT with the previously referred rules to predict the *foaf:knows* property for all the 172 users. With those results, we applied two distinct clustering techniques: the general purpose *k-means* clustering algorithm (Marques de Sá 2001),

and the *Markov Cluster Algorithm*[37] (MCA) (Van Dongen 2000), an unsupervised graph clustering algorithm.

After some initial experimentation, we defined the number of desired clusters in the k-means algorithm to 3, and the *inflation* property of the MCA to 1.6 (which also produces 3 clusters). Since the initialization of *cluster centroids* in k-means is random, the algorithm was run 100 times and the best solution is the one presented. Table 26 provides the cluster sizes and the number of shared members between solutions.

		K-means		
		K1 (114)	K2 (47)	K3 (11)
	C1 (135)	**102**	22	11
MCA	C2 (30)	5	**25**	0
	C3 (7)	7	0	**0**

Table 26. Link-based clustering analysis results. The table represents the number of shared members between the clusters of the two algorithms (e.g., cluster C2 and K2 share 25 individuals). Between brackets is the size of each cluster.

Even if the underlying techniques are conceptually distinct, both solutions provide similar clusters, both in size and composition. The biggest clusters from both solutions (*C1* and *K1*) are very similar, as well the second biggest clusters (*C2* and *K2*).

The Non-Relational Experiment

In the last experiments, we have seen how to classify and cluster relational domains. However, there are some domains that are "flat" by nature, i.e., they do not provide any meaningful relation between objects of the same kind. In this experiment, we study two of these domains: the Mushrooms dataset and the Titanic dataset.

Mushrooms Dataset

Based on the mushrooms dataset[38], we created a small ontology modelling the domain. The ontology is composed by 2 classes of mushrooms (*Edible* and *Poisonous*) and 6 properties (*hasCapColor*, *hasHabitat*, *hasOdor*, *hasSporePrintColor*, *hasStalkColorAboveRing*, and *hasStalkSurfaceBelowRing*). There are 8124 distinct mushrooms, and the task is to predict the class of the mushrooms given their

---

[37] http://micans.org/mcl/

[38] http://archive.ics.uci.edu/ml/datasets/Mushroom

properties. We splited the data in two disjoint sets, each one with 4062 mushrooms, and rotated their usage on the training and test of these rules:

Formula
$\forall x: !(hasOdor(x, Almond) \lor hasOdor(x, Anis) \lor hasOdor(x, None))$ $\Rightarrow Poisonous(x)$
$\forall x: hasSporePrintColor(x, Green) \Rightarrow Poisonous(x)$
$\forall x: hasOdor(x, None) \land hasStalkSurfaceBelowRing(x, Scaly)$ $\land !hasStalkColorAboveRing(x, Brown) \Rightarrow Poisonous(x)$
$\forall x: hasHabitat(x, Leaves) \land hasCapColor(x, White) \Rightarrow Poisonous(x)$

Table 27. Mushrooms dataset rules as provided in the dataset file. Rules for the *Edible* class are the negation of these four rules. Weights are not demonstrated since they vary with the training set.

Weights were learned generatively, and MC-SAT inference was performed. The (averaged) results can be seen on the next table.

Class	Specificity	Precision	Recall	F-measure
Edible (4208)	0.99	0.98	0.90	0.94
Poisonous (3916)	0.88	0.91	0.98	0.94
Weighted Avg	0.94	0.94	0.94	0.94

Table 28. Mushroom dataset classification results. Between brackets is the number of individuals in each class.

Since the rules were made by experts in the domain, they gave good results in the classification process.

Titanic Dataset

The Titanic Dataset[39] is composed by information about the passengers of the RMS Titanic ship that sunked in April 1912. There is information about the class (*First*, *Second*, *Third*, or *Crew*), age (*Adult* or *Child*), and sex of the passengers. The main task is to predict if a certain passenger survived to the accident given their properties. Using the dataset information, we created a simple ontology modelling the domain, with a total of 2201 passengers. Based on the famous *women-and-children-first* protocol that became famous in the Titanic disaster, we created three simple rules (Table 29).

---

[39] http://stats.math.uni-augsburg.de/Mondrian/Data/Titanic.txt

Formula
$\forall x : hasSex(x, Female) \Rightarrow Survivor(x)$
$\forall x : hasAge(x, Child) \Rightarrow Survivor(x)$
$\forall x : inClass(x, First) \Rightarrow Survivor(x)$

Table 29. Titanic rules. *Non-Survivor* rules are the negation of these three. Weights are not demonstrated since they vary with the training set.

The first two rules state that women and children could have more probability of being survivors, while the last rule states that passengers in first-class also could have more probabilities than the others of being saved. Using generative learning and MC-SAT, the data was split 50:50 in two disjoint sets, being the role of testing and training set swapped and the results averaged:

Class	Specificity	Precision	Recall	F-measure
Survivor (711)	0.81	0.61	0.60	0.61
NonSurvivor (1490)	0.60	0.81	0.81	0.81
Weighted Avg	0.67	0.75	0.75	0.75

Table 30. Titanic dataset classification results. Between brackets is the number of individuals in each class.

This results show that even with simple and easily comprehensible rules it is possible to achieve good classification results with Markov logic.

3.3. Probabilistic Reasoning by Learning Individuals/Probabilities

In the last section, we have explored the use of ontology individuals to automatically learn the uncertainty of the ontology axioms and perform inference with that information. This feature proved to be useful in domains where there was no information about that uncertainty, or in complex domains where this uncertainty is hard to infer, specially for humans. However, there are domains that are uncertain but do not have any type of information that could help us infer its uncertainty.

These domains are not uncertainty annotated, and do not have a sufficient number of individuals that allows learning the weights with some confidence in the results. In fact, in most of the cases, these domains do not have any individuals at all. In a preliminary study (Table 31) with 216 ontologies from the TONES ontology

repository[40], we found that approximately 75% of the ontologies do not have any type of individuals, and only about 7% had more individuals than formulas, fact that indicates a high probability of having a good number of individuals.

Number of ontologies	Number of ontologies with individuals	Number of ontologies with more individuals than formulas
216	54 (25%)	15 (7%)

Table 31. Preliminary study on ontology individuals.

This problem is mainly due to the fact that a large number of Semantic Web ontologies currently available were made to model pure terminological domains, with the main objective of answer questions about concepts and not individuals. In these ontologies, we have to find other ways of gathering information to learn the uncertainty of the axioms. In this thesis, we explored two approaches to tackle this problem: learn individuals and learn probabilities.

3.3.1. Learning Individuals

Due to the enormous quantity of textual resources currently avaliable, specially those present in the World Wide Web and avaliable through web search engines, extracting ontology individuals from those sources has became a task of growing interest. This is the task studied in the field of ontology population.

Ontology Population

The objective of *ontology population* techniques is to, given a source ontology and a corpus, extract individuals of that ontology from the corpus, with their class and property assertions. There are two main types of methods to perform this task: *supervised* and *semi-supervised* methods (e.g., (Tanev and Magnini 2006)), using machine learning classifiers to learn individuals; and *unsupervised methods*, mainly using pre-defined *lexico-syntactic patterns*. In this thesis, we explored unsupervised methods, mainly due to its unsupervised and domain-free applicability.

(Hearst 1992) defined 6 simple patterns to extract hyponymy relations from text. These patterns extract hyponyms by analisying expressions like "Animals such as dogs and cats" and "Dogs, cats, and other animals". As noted by several other works (e.g., (Evans and Street 2004) and (McDowell and Cafarella 2008)), these patterns

---

[40] http://owl.cs.manchester.ac.uk/repository/

can be also used to extract class assertions, since an hyponymy relation can be also seen as an assertion that a certain class or individual is a member of (i.e., a hyponym) and is subsumed by other class. Several other works expanded those patterns, by improving them or by proposing new patterns (e.g., (Etzioni et al. 2005) and (P. Cimiano, Ladwig, and Staab 2005)). (McDowell and Cafarella 2008) also propose several metrics to evaluate the ontology population results, and to choose the best class for an individual.

Proposed Approach

After exploring the results of the previously presented works, we implemented the following ontology individual lexico-syntactic extraction patterns:

Type	Pattern
Class Assertion	CLASS {,} such as {NP,}* {(and|or)} NP
	such CLASS as {NP,}* {(and|or)} NP
	NP {,NP}* {,} (and|or) {(all|every)} other CLASS
	CLASS {,} (including|{e}specially) {NP,}* {(or|and)} NP
	CLASS like {NP,}* {(and|or)} NP
	NP (is|are) {(a|an|the)} CLASS
Property Assertion	NP (is|are) {(a|an|the)} RELATION NP
	NP {,} RELATION NP

Table 32. Individual extraction patterns. CLASS and RELATION represent the class or relation that is to be populated, and NP represents a Noun Phrase. Optional elements are between braces, disjoint elements between parentheses, separated by a vertical bar. Asterisks indicate that optional elements can appear 0 or more times.

For example, if we want to populate the class "Animal", we look for expressions in the text like "Animals such as dogs, cats, and rabbits", "Lions are animals", "animals, specially chimpanzees and apes". For properties, for example "son of", we look for expressions like "Bob is the son of John" and "Bob, the son of John". *Noun Phrase* detection is performed with the following patterns:

Pattern
NP = {DT} {CD} {AP} NOMINAL
AP = {ADVERB} ADJECTIVE+
NOMINAL = NOUN+

Table 33. Noun phrase detection patterns. DT represents a determiner, CD a cardinal number, and AP an adjective phrase. Optional elements are between braces, and the plus indicates that the element must appear 1 or more times.

The ontology population procedure (Algorithm 1) was implemented in GATE[41], a framework for natural language processing that already provides a tokenizer, sentence detector, part-of-speech tagger, and morphological analyzer. Patterns were implemented in JAPE, a finite state transducer engine for annotations based on regular expressions, available in GATE.

Algorithm 1. Given an ontology *O* and a set of knowledge sources *KS*, the ontology population process is as follows:

1. Build Corpus
    a. If *KS* contains files
        i. Add files to corpus
    b. If *KS* contains search engines
        i. Issue search engine queries using patterns of Table 32, using the classes and properties of *O*
        ii. Add results title and snippet to corpus
2. Process Corpus
    a. Tokenization
    b. Sentence Splitting
    c. Part of Speech Tagging
    d. Morphological Analysis
3. Parse Results
    a. Apply Table 32 patterns to the processed corpus
    b. Match the found assertions with the properties and classes of *O*
    c. Add correct assertions to *O*

As knowledge sources, we can use electronic documents (e.g., *Microsoft Word* and *Adobe PDF* documents) or web search engines. We implemented the access to the three most used web search engines (*Google*, *Yahoo*, and *Live Search*). However, each of these web search engines provides distinct limitations (Table 34), which can influence the quality of the extracted individuals. Due to the maximum number of

---

[41] http://gate.ac.uk

queries restriction of Live Search, in this work we mainly use Google and Yahoo search engines capabilities.

Web Search Engine	Max. Number of Queries	N. Results per query	Registration
Google Search API[42]	Unlimited*	64 (8 per time)	Optional
Yahoo BOSS[43]	Unlimited*	100	Mandatory
Live Search API 2.0[44]	7/second	1000 (50 per time)	Mandatory

Table 34. Web search engines APIS and their features. The unlimited number of queries of Google and Yahoo is theoretical.

Example

Suppose we want to populate a simple ontology about the characters of the animation television series *Dragon Ball*[45] and their fights (Figure 18). Using the extraction rules previously defined, and using the Google Search API as the knowledge source with 8 results per query, we found the assertions of Table 35.

From the 11 Dragon Ball characters found, only one, "character" is wrong. This is due to the phrase "its *characters are dragon ball characters*". Some of them, like *Kid Chi* and *Tien* are abbreviations of the full names (*Kid Chi-Chi* and *Tien Shinhan*, respectively).
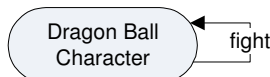
Figure 18. Dragon Ball ontology. Directed edge represents a property, rounded square a class.

---

[42] http://code.google.com/apis/ajaxsearch/

[43] http://developer.yahoo.com/search/boss/

[44] http://dev.live.com/livesearch/

[45] http://en.wikipedia.org/wiki/Dragon_Ball

Class/Property	Assertion	Count
	goku	4
	piccolo	3
	frieza	2
	emperor pilaf	1
	character	1
Dragon Ball Character	kid chi	1
	majin buu	1
	tien	1
	vegeta	1
	king piccolo	1
	chiaotzu	1
	(goku,piccolo)	1
	(piccolo,frieza)	1
fight	(goku,tien)	1
	(vegeta,frieza)	1

Table 35. Dragon Ball ontology population individuals.

Experimentation

In this experiment, we automatically learn an ontology (Figure 19) about diseases and their symptoms and perform clustering on it.
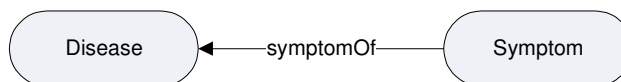
Figure 19. Diseases ontology. Directed edge represents a property, rounded squares classes.

The first step is to learn the individuals of the class *Disease* using the class assertion patterns of Table 32. As corpus, we used both Google Search and Yahoo BOSS APIs with the maximum results possible per query (64 for Google, 100 for Yahoo). As we can see in Table 36, both search engines give similar precision results. Yahoo BOSS gives a bigger total number of individuals due to the bigger number of results per query. Precision results could be improved if we ignored the diseases that appeared only once in the corpus. However, their number is largely reduced (to about one third in both search engines). After joining all the correct diseases found by both search engines, and removing a small number of non-human diseases like powdery mildew, a total of 271 distinct diseases were found.

Web Search Engine	Individuals	Total	Correct	Precision
Yahoo BOSS	All	330	231	0.70
	More than 1 count	117	92	0.79
Google Search API	All	193	128	0.66
	More than 1 count	67	51	0.76

Table 36. *Disease* ontology class population results.

The next step is to learn the symptoms of those diseases. Using the property assertions patterns of Table 32, we queried both search engines with two textual variations of the *symptomOf* property: "symptom of" and "sign of". Since we already know which diseases we want to find symptoms, the last part of the patterns can be easily substituted by the desired disease, i.e., we can query for "* is a symptom of Malaria" and "* is a sign of Malaria" instead of the more general queries "* is a symptom of *" and "* is a signal of *". The results (Table 37) indicate similar precision values for both search engines.

After joining all the correct symptom assertions produced by both search engines, and removing the diseases without symptoms, we had an ontology composed by 140 diseases, 459 symptoms, and 671 symptoms assertions. One interesting fact is that there are individuals that are represented as both diseases and symptoms. For example, *Bronchitis* is defined as a disease, but also as a symptom for other diseases, like *Lung Cancer*. This fact occurs with 21 of the 140 diseases.

Web Search Engine	Individuals	Total	Correct	Precision
Yahoo BOSS	All	618	389	0.63
	More than 1 count	228	144	0.63
Google Search API	All	661	419	0.63
	More than 1 count	187	133	0.71

Table 37. *SymptomOf* ontology property population results.

An interesting task to perform with this kind of ontology is to create clusters of similar diseases. For this purpose, we can use a set of similar rules to those used in the clustering experimentation of Section 3.2.1:

Weight	Formula
0.21	$\forall x, y, z : symptomOf(z, x) \wedge symptomOf(z, y) \Rightarrow Similar(x, y)$
7.27	$\forall x, y : Similar(x, y) \Leftrightarrow Similar(y, x)$
0	$\forall x, y, z : Similar(x, y) \wedge Similar(y, z) \Rightarrow Similar(x, z)$

Table 38. Diseases clustering rules.

In this case, the first rule defines that two diseases are similar if they share symptoms. The other two rules define *Similar* as a *symmetric* and *transitive* property, respectively. Weights were learned generatively using all the individuals available. We used MC-SAT with the previously referred rules to predict the *Similar* property for all the 140 diseases. With those results, we applied two partitional

clustering techniques (Zhao and Karypis 2005) available in the *CLUTO*[46] toolkit: *Repetead bisections* (RB) and *Nearest-Neighbor Graphs* (NNG), both configured to create 5 clusters each:

		RB				
		K1 (84)	K2 (15)	K3 (24)	K4 (8)	K5 (9)
	C1 (42)	**42**	0	0	0	0
	C2 (20)	7	**13**	0	0	0
NNG	C3 (35)	3	0	**23**	0	9
	C4 (29)	20	0	1	**8**	0
	C5 (14)	12	2	0	0	**0**

Table 39. Link-based clustering analysis results. The table represents the number of shared members between the clusters of the two algorithms (e.g., cluster C2 and K2 share 13 individuals). Between brackets is the size of each cluster.

Both solutions created similar clusters, specially (*C2*, *K2*) and (*C3*, *K3*). Some of these clusters can be analyzed by their main symptoms. For example, all the diseases in cluster *K5* have a common symptom: depression. This cluster includes diseases like migraines, anxiety, alzheimers, and bipolar disorder. Cluster *K4* is composed by diseases related to the respiratory system, like lung cancer, tuberculosis, asthma, pneumonia, and diphteria. These diseases share symptoms like coughing, chest pain, and bronchodilation. In Cluster *C5*, most of the diseases, like cholera, salmonella, and colon cancer, share diarrhea as a common symptom.

3.3.2. Learning Probabilities

An OWL2 ontology is composed by a set of axioms that model the semantic relations between entities of the domain. In the last two sections, we have seen how to use individuals to learn automatically the uncertainty of those axioms. However, these approaches can have some problems:

- We need a representative number of individuals to perform the learning process with some confidence in the results. This arise problems when: we do not have the desired number of individuals; or the final number of individuals is too large to perform weight learning efficiently.

- Weights are hard for humans to understand, not only because they are not normalized, but also because in Markov logic they do not have a direct

---

[46] http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview

correspondence to probabilities, and therefore their values can be misleading.

An interesting approach would be to automatically learn the probability of axioms, instead of weights, and without the need for learning individuals. In this thesis, we explore the use of semantic similarity techniques to perform this task.

Semantic Similarity

Semantic similarity is the process of finding the similarity between two words or entities. This is usually done by studying the *co-occurrence* between those words or entities in a textual corpus: if they appear together many times, this could be the indication that they are somehow semantically related. The most used techniques use the redundancy and size of a huge corpus, like the World Wide Web, and the results of search engines to measure that similarity. This is usually done by counting the number of results returned by those search engines in specific queries related to the words or entities whose similarity we want to assert.

In this thesis, we explored 6 distinct metrics to perform this task (Bollegala, Matsuo, and Ishizuka 2007) (Church and Hanks 1990) (Magnini et al. 2002) (Cilibrasi and Vitanyi 2004):

$$WebJaccard(P,Q) = \frac{H(P,Q)}{H(P) + H(Q) - H(P,Q)} \tag{3.9}$$

$$WebOverlap(P,Q) = \frac{H(P,Q)}{\min(H(P), H(Q))} \tag{3.10}$$

$$WebDice(P,Q) = \frac{2H(P,Q)}{H(P) + H(Q)} \tag{3.11}$$

$$PointwiseMutualInformation\,(P,Q) = \log_2\left(\frac{H(P,Q)}{H(P)H(Q)} * N\right) \tag{3.12}$$

$$CorrectedConditionalProbability\,(P,Q) = \frac{H(P,Q)}{H(P)H(Q)^{\frac{2}{3}}} * N^{\frac{2}{3}} \tag{3.13}$$

$$NormalizedWebDistance(P,Q) = \frac{\max(\log H(P), \log H(Q)) - \log H(P,Q)}{\log N - \min(\log H(P), \log H(Q))} \tag{3.14}$$

Here
- $H(P)$ is the number of search engine results for search phrase P
- $H(Q)$ is the number of search engine results for search phrase Q
- $H(P,Q)$ is the number of search engine results for the tuple of search phrases PQ
- N is the total number of indexed pages by the search engine

When search phrases P or Q are composed by more than one word, they are enclosed by quotation marks. The index size of search engines, N, is usually not

known, but it can be approximated by using sampling techniques (e.g. (Bar-Yossef and Gurevich 2008)). Based on the updated results provided by (de Kunder 2009), in this work we estimate the size of the Google index to 40 billion pages, and Yahoo to 20 billion.

WebJaccard, WebOverlap, and WebDice give a value in the interval [0,1], with higher values representing bigger similarities. Pointwise Mutual Information (PMI) and Corrected Conditional Probability (CCP) give a real positive number, with higher values also representing bigger similarities. The Normalized Web Distance (NWD) gives a positive real value, with the values closer to 0 representing the most similarity. However, some of these properties can change in some rare conditions (e.g., when search engines return more results in $H(P,Q)$ than in $H(P)$ and/or $H(Q)$). In these cases, if the result diverges from its range, we can clip the value to the closer value of the range (e.g., if PMI gives a result below 0, we can simply set it to 0).

Example

To assert the semantic similarity between the words "dog" and "pet" using the Google Search API, we perform the following queries:

Query	Result Count
dog	64,900,000
pet	61,200,000
dog pet	53,500,000

Table 40. Google Search API query results count.

The metrics results can be seen on Table 41, accompanied by other examples.

Similarity	WebJaccard	WebOverlap	WebDice	PMI	CCP	NWD
(dog,pet)	0.74	0.87	0.85	9.07	62.08	0.03
(coimbra,portugal)	0.01	0.45	0.02	6.95	18.97	0.48
(google,Pedro)	0.00	0.20	0.01	1.69	0.31	0.83
(good,evil)	0.12	0.83	0.21	6.94	12.27	0.30
(banana, einstein)	0.00	0.01	0.01	4.80	1.65	0.61

Table 41. Examples of semantic similarity metrics results using the Google Search API.

Proposed Approach

The presented techniques assert the semantic similarity between any two entities by giving a confidence value to the likelihood that these entities have any type of semantic relation. However, in our specific case, we do not want to calculate if two entities are connected by any semantic relation: we already know they probably are, we just want to give a confidence value to that relation. For example, if we have the axiom $SubClassOf(Dog, Pet)$, we are not interested in asserting if the entity *Dog* has any type of semantic relation with *Pet*: we already know that they probably have one

(a subsumption relation). What we want is a confidence value to that specific relation between those two concrete entities. For this purpose, we can use the previously referred metrics with some modifications, namely in the format of the search engine queries.

If we have a semantic relation defined by (P, R, Q), where P is the subject, Q the object, and R the relation, we redefine the previous query definitions as:

- $H(P)$ is the number of search engine results for search phrase "$PR*$"

- $H(Q)$ is the number of search engine results for the search phrase "$*RQ$"

- $H(P, Q)$ is the number of search engine results for the triple of search phrases "PRQ"

Here, $*$ represents a web search engine wildcard that matches any potential word. Note that query phrases are obligatorily enclosed with quotation marks.

Example

If we have the axiom $SubClassOf(Dog, Pet)$, which can be translated to the semantic relation $(Dog, is\ a, Pet)$, we perform the following queries using Google Search API:

Query	Result Count
"dog is a *"	287,000
"* is a pet"	182,000
"dog is a pet"	785

Table 42. Google Search API query results count.

The metrics results, with some more examples, can be seen on the next table.

Relation	WebJaccard	WebOverlap	WebDice	PMI	CCP	NWD
(dog,is a,pet)	0.002	0.004	0.003	9.230	9.949	0.480
(Obama,president of,United States)	0.031	0.135	0.059	16.745	1,175.085	0.220
(Einstein, eat,banana)	0.000	0.000	0.000	0.000	0.000	+∞
(Coimbra,is in,Portugal)	0.000	0.003	0.000	12.432	44.438	0.536
(good,is,evil)	0.000	0.006	0.000	2.891	0.180	0.820

Table 43. Examples of the modified semantic similarity metrics results using the Google Search API.

As we can see, the first 3 metrics give results always near 0. This is mainly due to the fact that $H(P, Q)$ is constantly small compared to $H(P)$ and $H(Q)$, and since in these metrics $H(P, Q)$ is divided by some form of combination between $H(P)$ and $H(Q)$, the resulting value is small. The other metrics take the size of the index, N, into account, and, based on our common sense, give better results.

Given those results, we have metrics that give values in the range [0,1], but in this case they always give values near 0. And we have metrics that work as intended, but

give a positive real value. However, we need that those metrics gave us a well distributed value between [0,1] that could be interpreted as our confidence on the veracity of the relation. To this purpose, we need to normalize those values. If we have a set of values $[v_0,...,v_n]$, a value v can be normalized into range $[x, y]$ by performing a linear scaling transformation:

$$v' = \frac{v - min}{max - min} * \big((y - x) + x\big).$$

3.15

Since we know that $x = 0$, $y = 1$, and $min = 0$ in all of the metrics, this formula can be simplified into a simple division by max:

$$v' = \frac{v}{max}$$

3.16

In the case of NWD, this value must be also converted into a dissimilarity:

$$v'' = 1 - v'$$

3.17

If we apply the normalization to the previous results, we get the following values:

Relation	WebJaccard	WebOverlap	WebDice	PMI	CCP	NWD
(dog,is a,pet)	0.055	0.032	0.056	0.551	0.008	0.995
(Obama,president of,United States)	1.000	1.000	1.000	1.000	1.000	0.998
(Einstein,eat,banana)	0.000	0.000	0.000	0.000	0.000	0.000
(Coimbra,is in,Portugal)	0.002	0.021	0.002	0.742	0.038	0.995
(good,is,evil)	0.003	0.042	0.004	0.173	0.000	0.992

Table 44. Normalization of the metrics values in Table 43.

In some cases, a semantic relation can have more than one query pattern. For example, the *subClassOf* relation can have all the patterns defined in Table 32, since these patterns where originally designed to infer *subClassOf* relations. In those cases, for one semantic relation, we have several values for each metric. The final metric value can be simply an arithmetic mean of those values, e.g., if $PMI(R) = \{v_1, ..., v_n\}$, the new PMI is:

$$PMI'^{(R)} = \frac{\sum_i v_i}{count\big(WebPMI(R)\big)}$$

3.18

where $count\big(PMI(R)\big)$ is the number of patterns used for semantic relation R.

Experimentation

In Section 3.1.2, we have extracted a taxonomy about animals from a web search engine, and used a simple metric based on the frequency of the extracted entities to assert the confidence value of the relations. In this experiment, we use the presented metrics to assert those confidence values. Using the patterns of Table 32, we issued the queries of Table 45 and used the Google Search API to calculate the various metric values. The results can be seen in Table 46.

Pattern	$H(P)$	$H(Q)$	$H(P,Q)$
1	"* such as P"	"Q such as *"	"Q such as P"
2	"such * as P"	"such Q as *"	"such Q as P"
3	"P (and OR or) (all OR every)? other *"	"* (and OR or) (all OR every)? other Q"	"P (and OR or) (all OR every)? other Q"
4	"* (including OR specially OR especially) P"	"Q (including OR specially OR especially) *"	"Q (including OR specially OR especially) P"
5	"* like P"	"Q like *"	"Q like P"
6	"P (is OR are) (a OR an OR the)? *"	"* (is OR are) (a OR an OR the)? Q"	"P (is OR are) (a OR an OR the)? Q"

Table 45. Search engine queries based on patterns from Table 32. ? indicates an optional pattern, indicating that two queries will be issued: one with that pattern, and one without.

Relation	Desired	WebJaccard	WebOverlap	WebDice	PMI	CCP	NWD
rabbits, animals	1	0.27	0.81	0.28	0.96	0.53	0.99
dogs, animals	1	1.00	1.00	1.00	0.98	0.82	1.00
horses, animals	1	0.63	0.96	0.65	1.00	0.77	1.00
lions, animals	1	0.16	0.60	0.17	0.94	0.55	0.99
house rabbits, rabbits	1	0.00	0.18	0.00	0.28	1.00	0.00
cats, rabbits	0	0.01	0.21	0.01	0.48	0.02	0.66
cats, dogs	0	0.11	0.09	0.11	0.62	0.09	0.83
arthritis, dogs	0	0.00	0.00	0.00	0.19	0.00	0.33
lusitano, horses	1	0.00	0.08	0.00	0.26	0.33	0.00
Correlation	1	0.43	0.62	0.44	0.45	0.85	0.07

Table 46. Modified semantic similarity metrics results based on query results from Table 45.

CCP is the metric with best correlation. This result is mainly derived from the fact that it gives very low values to the wrong assertions (i.e., the ones with 0 as its desired value). The analysis of the results of the appliance of the individuals patterns (Table 47) demonstrate that there is no single pattern that is responsible for the high correlation of CCP: some patterns are good to measure specific assertions (e.g. pattern 4 and 6 to assertion *(House Rabbits, Rabbits)*), while others give average results in all the patterns (e.g. pattern 1). The combination of all the patterns is the key to the good results.

The other metrics also give good results, especially WebOverlap. NWD contains a low correlation since it can only approximate the correct value of the four first assertions.

Relation	Desired	1	2	3	4	5	6
rabbits, animals	1	0.54	0.58	0.26	0.08	1.00	0.39
dogs, animals	1	0.48	0.51	1.00	0.12	0.61	0.33
horses, animals	1	0.46	0.26	0.97	0.14	0.62	1.00
lions, animals	1	0.26	1.00	0.32	0.09	0.81	0.17
house rabbits, rabbits	1	0.00	0.00	0.00	1.00	0.00	0.93
cats, rabbits	0	0.00	0.00	0.05	0.00	0.06	0.05
cats, dogs	0	0.00	0.00	0.21	0.00	0.07	0.11
arthritis, dogs	0	0.00	0.00	0.00	0.00	0.00	0.00
lusitano, horses	1	1.00	0.00	0.00	0.00	0.00	0.84
Correlation	1	0.65	0.54	0.42	0.37	0.58	0.71

Table 47. Results of the modified CCP metric, with separate patterns from Table 45.

3.4. Final Remarks

The main issue addressed in this chapter is: how can an OWL2 ontology be transformed into a MLN so we can use Markov logic capabilities? As seen, MLNs formulas can be acquired by interpreting ontology's axioms as first-order logic formulas. For the MLN weights, several approaches were proposed: interpret ontology's uncertainty annotations as weights; use ontology individuals to learn the weights; or learn probabilities and individuals of the ontology and use them to learn the weights.

As seen, each of these approaches has its advantages and limitations, being appropriate for special situations. By analyzing them, two conclusions can be drawn. The first is that the more complex they are, the more probable it is that the quality of results declines. For example, if we have an ontology already with individuals, and we learn the weights using those individuals, the results are probably better than by learning the weights using individuals learned by a web search engine. This result leads to the second result, that the less information we have about the ontology, the worst will be the quality of the results. For example, having an ontology with both uncertainty annotations and individuals is better than have the same ontology only with individuals. Both facts must be taken in account when choosing any of the proposed approaches.

In the next chapter, we present *Incerto*, the system that was used to perform all the experimentations in this chapter.

4. Incerto – A probabilistic Reasoner for the Semantic Web

Using the ideas of the previous chapter, we developed *Incerto*, a probabilistic reasoner for the Semantic Web based on Markov logic. The system was developed in Java, and is freely available through a GNU Lesser General Public License[47] (LGPL) at http://code.google.com/p/incerto. This section describes the features and functionalities, requirements, and architecture of the system. The organization of the code, some usage examples, and scalability tests are also provided.

4.1. Features and Functionalities

The main features and functionalities of the system are:
- **Load ontologies with or without uncertainty annotations.** The system must read annotated ontologies (with weights or probabilities) and ontologies without annotations.
- **Selection of ontology axioms.** The system must provide means to select subsets of the ontology to perform the reasoning.
- **Weight learning.** The system must provide mechanism to learn the weights automatically.
- **Parameterization of algorithms.** The system must allow the change of the parameters of the main algorithms.
- **Programmatic, command line, and visual access.** The system must provide programmatic access, a command line interface, and a visual interface.

4.2. Requirements

In this section, we define the system's requirements. These are divided in *functional requirements*, describing the system from the user's perspective, and *non-functional requirements*, describing required properties and constraints of the system.

---

[47] http://www.gnu.org/copyleft/lesser.html

The functional requirements of the system are:
- **Probabilistic reasoning using Markov logic in OWL2 ontologies.** Given a valid OWL2 ontology, the system must be able to perform probabilistic reasoning on it.
- **Intuitive result representation.** The reasoning results must be presented in an intuitive and understandable way.

The non-functional requirements, related to the usability, performance, and supportability of the system, are:
- **Simple access and installation**. Novice users must be allowed to install and operate with the system with little or no training. However, the system must be also prepared to deal with the needs of more experienced users.
- **High performance**. Since probabilistic reasoning is a very hard task in general, the system must be optimized to our specific domain. The parameters of performance definition of the system should be also visible to the user.
- **Easy modification and expansion.** The system shall allow users to easily modify or expand the current system to their concrete needs.

4.3. Architecture

In this section, we present the system's architecture (Figure 20) and describe its main components. There are two main components in the system: the *System Core*, responsible for the processes related to the reasoning, and the *Interface Layer*, responsible for the interaction with the user.
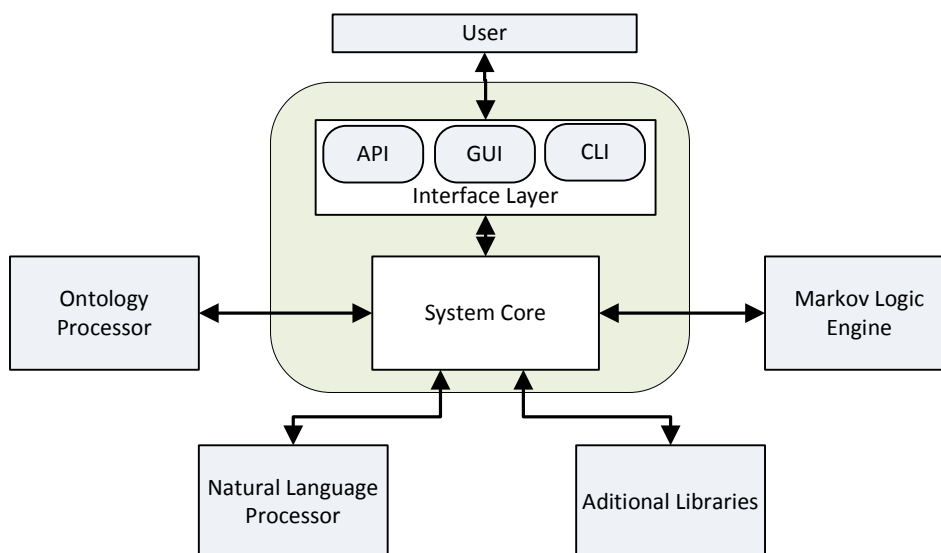
Figure 20. System architecture.

4.3.1. System Core

The *System Core* is the main component of the system. It interacts with four main external components: the *Ontology Processor*, responsible for the reading, processing, and writing of OWL2 ontologies; the *Markov Logic Engine*, responsible for the Markov logic reasoning and learning processes; the *Natural Language Processor*, responsible for the processing of natural language resources; and *Additional Libraries*, composed by a set of general libraries used in several minor tasks. Next, each of these components will be explained.

Ontology Processor

Ontology processing is provided by *OWL API*[48] (Horridge, S. Bechhofer, and Noppens 2007), a Java open-source tool to process OWL ontologies. It provides a simple and easy Application Programming Interface (API) to deal with OWL ontologies, providing fast and efficient in-memory ontology processing. The main difference of *OWL API* to others ontology processors, like *Jena*[49], is that it uses a functional syntax representation, representing ontological knowledge as *axioms* instead of triples. This way, a much cleaner and easier to understand representation is achieved, providing a frame style modeling, easier to access programmatically.

Markov Logic Engine

Markov logic capabilities can be provided by two distinct packages: Alchemy and PyMLNs.

Alchemy

Alchemy[50] (S. Kok et al. 2007) is a software package providing a series of algorithms for statistical relational learning and probabilistic logic inference, based on the Markov logic representation. It provides a series of state of the art algorithms for inference and learning in Markov logic (Table 48).

---

[48] http://owlapi.sourceforge.net/

[49] http://jena.sourceforge.net/

[50] http://alchemy.cs.washington.edu/

Feature	Algorithms
Weight Learning	Generative and Discriminative (Voted Perceptron, Conjugate Gradient, and Newton's Method)
Structure Learning	Top-Down Learning
Inference	MAP/MPE inference and Probabilistic inference (MC-SAT, Gibbs Sampling, Simulated Tempering, and (lifted) Belief Propagation)

Table 48. Alchemy features and algorithms.

Other interesting features, like support of continuous domains and online inference are also supported. The application is developed in *C++*, being provided both an API and a console interface. Currently, it only supports natively *Unix* systems, but a porting to *Windows* systems with *Cygwin*[51] is possible.

PyMLNs

PyMLNs[52] is a Python toolkit to work with Markov logic networks. It provides a Python implementation of many algorithms for inference and learning in Markov logic:

Feature	Algorithms
Weight Learning	Maximum likelihood (Log-likelihood and Pseudo-log-likelihood)
Inference	Probabilistic inference (Exact Inference, MC-SAT, and Gibbs Sampling)

Table 49. PyMLNs features and algorithms.

Some interesting features, like constraints on formulas probabilities and support for domains where the weights are in the [0,1] interval, are also provided. *PyMLNs* also provides a graphical user interface to work with Markov logic, using Alchemy or the intern engine as the Markov logic engine.

Natural Language Processor

Natural language processing is provided by the <u>G</u>eneral <u>A</u>rchitecture for <u>T</u>ext <u>E</u>ngineering[53] (GATE) (Cunningham et al. 2002). GATE provides a comprehensive Java

---

[51] http://www.cygwin.com/

[52] http://www9.cs.tum.edu/people/jain/mlns/

framework for human language processing. It is composed by several components and plugins that cover a large portion of the natural language processing tasks. In this work, we used the following components:

- **ANNIE**, <u>A</u> <u>N</u>early-<u>N</u>ew <u>I</u>nformation <u>E</u>xtraction system, which provides several components necessary to information extraction tasks, like tokenizers, sentence splitters, and a Part of Speech tagger;

- **JAPE,** the <u>J</u>ava <u>A</u>nnotation <u>P</u>atterns <u>E</u>ngine, a regular expression based language to parse annotations generated by other components (e.g. ANNIE). JAPE grammars are composed by a set of pattern/action rules that are applied to annotated documents;

- **GATE Morphological Analyzer** identifies the lemma, i.e., the canonical form of a word, and affix of a token identified with a part of speech tag. Morphological analysis is performed using a set regular expression rules.

Additional Libraries

The System core also uses a set of additional libraries:

- **ANTLR**[54], <u>AN</u>other <u>T</u>ool for <u>L</u>anguage <u>R</u>ecognition, is a language tool that provides a Java framework for constructing recognizers, compilers, and translators from grammatical descriptions containing actions. In this work, it is used to parse the output files from the Markov logic engines;

- **JSON**[55], the <u>J</u>ava<u>S</u>cript <u>O</u>bject <u>N</u>otation, is a lightweight data-interchange format. It provides a simple language based on lists and name/value pairs to exchange data. In this work, it is used to communicate with external web services, like those provided by web search engines.

System Behavior

The general function of the system is defined in Figure 21. In the *Process Ontology* process, a valid OWL2 ontology is transformed in an efficient computational representation using an *Ontology Processor*. In this phase, the ontology is also transformed in first-order logic. If the ontology does not contain uncertainty

---

[53] http://gate.ac.uk/

[54] http://www.antlr.org/

[55] http://www.json.org/

annotations, or those uncertainty annotations are not weights, we have to learn those weights using the techniques of Sections 3.2 and 3.3. In the *Reasoning* process, the system has all the information needed for reasoning: a set of weighted formulas (i.e., a Markov logic network) and a query. Using a *Markov Logic Engine*, the reasoning results are returned to the interface layer.
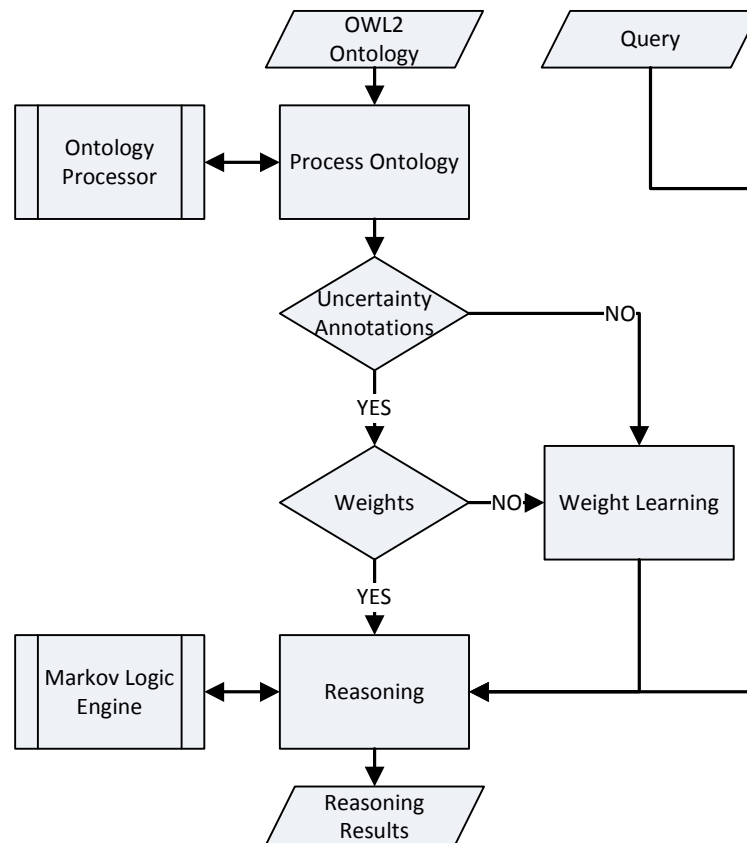
Figure 21. Flow chart representing the behavior of the System Core.

4.3.2. Interface Layer

The interface layer is responsible for the interaction with the user. There are three ways of communicating with the system:

- **Application Programming Interface (API)**, providing a programmatic access to the system. This allows the incorporation of the system in other applications.
- **Graphical User Interface (GUI)**, providing a visual access to the system. This allows the system to be intuitively used by the user.

- **Command-line Interface (CLI)**, providing a command-based access to the system. CLI capabilities are provided by JewelCLI[56], an annotated interface based library to parse and present command line arguments.

Due to the complexity of some operations, the API is the only interface that provides all the functionalities of the system. The other interfaces only provide the basic functionalities, sufficient for most of the tasks.

4.4. Code Organization

The code is organized using a well defined package hierarchy (Figure 22). The function of the main packages can be easily described:
- **Engines** –Alchemy reasoning engines wrappers and utilities
- **Exceptions** – Exceptions that can be generated by the system (e.g. *MarkovLogicEngineException*)
- **Experiments** – Implementation of the experiments used in this thesis
- **FirstOrderLogic** – Classes related to first-order logic (FOL)
    - ○ **Model** – Classes representing FOL elements (e.g. *Formula*, *Predicate*, *Variable*)
        - ▪ **Visitors** – Visitor design pattern implementations for printing *Model* elements
    - ○ **Parser** – Classes related to FOL file parsing
- **Interfaces** – Implementations of the Interface Layer
    - ○ **API** – API Interface Layer
    - ○ **CLI** – CLI Interface Layer
    - ○ **GUI** – GUI Interface Layer
- **Learners** – Systems and utilities used to learn ontologies and their elements from textual resources (e.g., *GateTaxonomyLearner*, *GateEvidenceLearner*)
    - ○ **OntologyPopulation** – Classes related to the learning of ontology individuals
        - ▪ **Model** – Classes representing elements used in the *OntologyPopulation* tasks (e.g., *NounPhrase*, *TextualEntity*)
    - ○ **Sources** – Sources used by learners (e.g. *DiskFiles*, *SearchEngine*)

---

[56] http://jewelcli.sourceforge.net/

- **SearchEngines** – Implementation of interfaces for web search engines (e.g. *GoogleSearchAPI*, *YahooBossAPI*)
- **Model** – Classes representing general system entities (e.g. *MLN, Query*)
- **Onto** – Classes and utilities related to Ontology Processing
- **Test** – Examples of system applications
- **Utils** – General system utilities (e.g., *InOutUtils*, *MathUtils*, *SystemUtils*)
  - **Settings** – Utilities related to the read and parse of external settings files
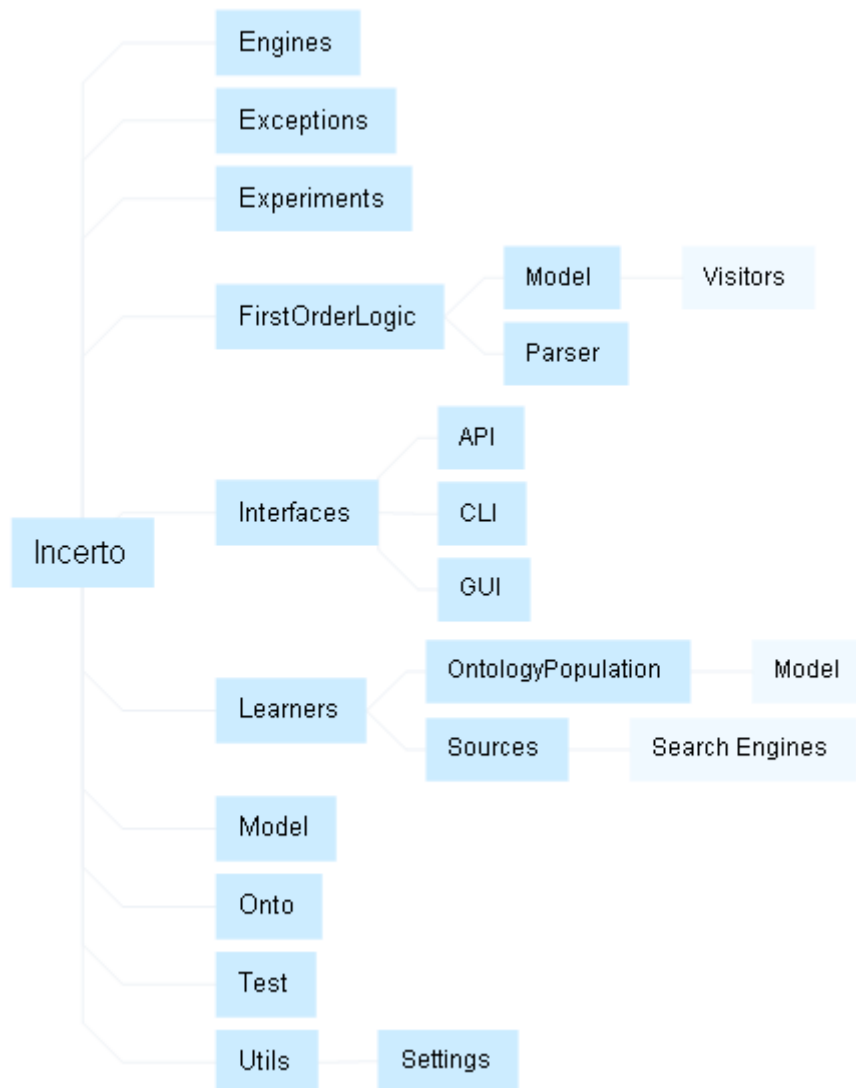
Figure 22. Code packages hierarchy.

4.5.   Usage Example

In this section, we provide a simple usage example of *Incerto*. Suppose we have an OWL2 ontology, *ontology.owl*, that contains various individuals. Our task is to use

those individuals to learn the uncertainty of the ontologies' axioms, and query for the conditional probabilities of individuals belong to class *Query*. This task can be easily performed using any of the Incerto's interfaces.

API

The API provides a simple access to the all the utilities developed in this thesis. For this usage example, seven lines of code were needed:

```
1.   MLN onto = new parserOWLAPI().onto2MLN("ontology.owl");
2.   MarkovLogicEngine engine = IncertoSettings.getInstance().ML_ENGINE;
3.   Query q = Query.parseQuery("Query");
4.   Evidence e = new Evidence(onto.getEvidences());
5.   MLN mln = engine.weightlearning(onto,e);
6.   ReasoningResults res = engine.inference(mln, e, q);
7.   System.out.println(res);
```

In the first two lines, the ontology is transformed in a MLN, and the reference for the default Markov logic engine is acquired. In lines 3 and 4, the desired query and the evidence individuals (in this case, all the individuals of the ontology) are created. In line 5, weight learning is performed, while in line 6 the learned MLN is used to perform inference. The results are printed in line 7.

GUI

The GUI provides the basic functionalities of *Incerto*. For this usage example, four steps were needed (Figure 23):
1. Load the ontology
2. Start the weight learning process
3. Insert the value "Query" in the *Query* property
4. Start the inference

After performing these steps, inference results are shown in a new window.

CLI

The console interface is composed by several commands[57]:
- **<-s <string>>** Source ontology location.

---

[57] Required commands are between angle brackets <>, while optional commands are between square brackets [].

- **[-l <string>]** Ontology with individuals to learn the weights. If omitted, the individuals of the source ontology are used for this task.
- **[-i <string>]** Ontology with individuals to be used in the inference process. If omitted, the individuals of the source ontology are used for this task.
- **[-r <string>]** Results file location. If omitted, results are written to the console.
- **<-q <string>>** Query predicates. They can be composed by the name of a class or property (e.g., *Person;isFatherOf*) with (optional) additional information about individuals (e.g., *Person(Pedro);isFatherOf(x,Pedro)*). Multiple query predicates can be added, separated by a semicolon (e.g., *Person;Person(Pedro);isFatherOf;isFatherOf(x,Pedro)* is a valid query). Free variables must be represented as lowercase digits.
- **[-e <string>]** Extra first-order logic rules file location.
- **[--disablewl]** Disable weight learning.
- **[--mla]** If set, instead of the probabilities, the most likely assignments to all query atoms are returned.
- **[--help]** Help.

In this usage example, a simple command was necessary:

```
java –jar incerto.jar –s ontology.owl –q Query
```
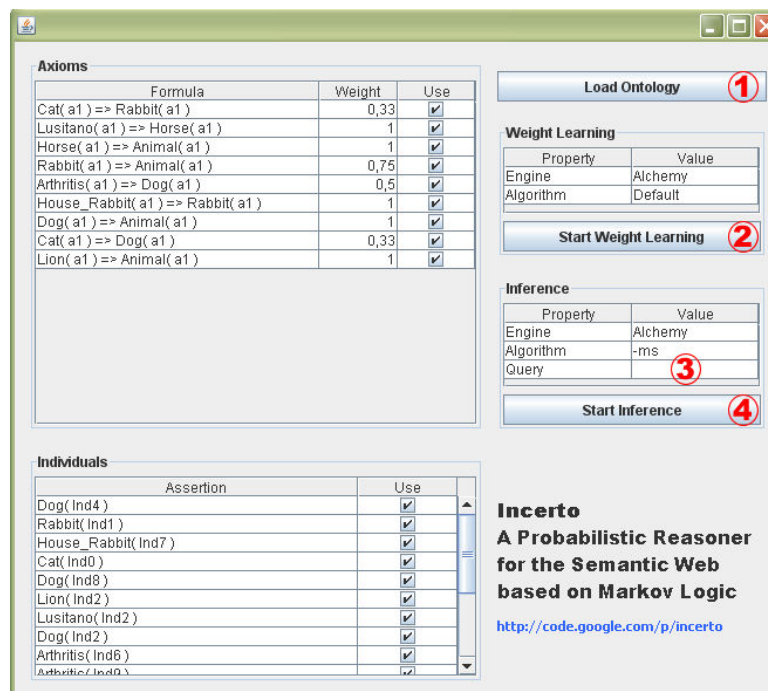

Figure 23. Graphical User Interface usage example.

4.6. Scalability Tests

In this section, we study the scalability of Markov logic procedures in the Semantic Web domain, as implemented in *Incerto*. Our purpose is to measure the scalability of three distinct procedures:

- *Pre-processing*, composed by the load and transformation of ontologies in MLNs;
- *Weight Learning*, using the generative algorithm (see Section 2.5.3);
- *Inference,* using the MC-SAT algorithm (see Section 2.5.2).

For this purpose, those procedures were performed on seven distinct ontologies, each one with a varied number of individuals. This variation was made by randomly populating each ontology with a set of individuals (we tested with 1, 10, 100, 1.000, and 10.000 individuals), using these individuals to make an average of three assertions for each ontology class or property. The following ontologies were used:

- **Animals** – Automatically learned ontology (Figure 13), composed by 9 classes and a simple taxonomic structure;
- **Substances** – Automatically learned ontology (Figure 14), composed by 20 classes and a more complex taxonomic structure;
- **Body Gestures** – Simple ontology about body gestures (Table 14), with 13 classes;
- **Diseases** – Ontology about diseases and their symptoms (Figure 19). The ontology was augmented with the rules of Table 38, being composed by three classes and one property.
- **Social Network** – Social network of Section 3.2.1, with three classes and three properties. Augmented with the rules of Table 22 and Table 24.
- **GoldDLP** – Financial ontology used in Section 3.2.1, with 39 classes and 6 properties.
- **Wine** – Ontology about wines[58], composed by 43 classes and 13 properties.

The averaged results (5 runs for each experiment), using an *Intel Centrino Duo T2300* with 1536 MB of memory and the Alchemy engine, can be seen on Figure 24, Figure 25, and Figure 26. Some interesting results were found:

- Pre-processing takes a relatively small time comparing to the weight learning and inference procedures, especially when the number of individuals is high.

---

[58] http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine

The *Wine* ontology took more time in this procedure, mainly because it contains more complex axioms, making its interpretation to first-order logic more difficult;

- *Animals*, *Substances*, and *Body Gestures* ontologies take less than six seconds to perform weight learning with 10.000 individuals. However, *Diseases*, *Social Network*, and *GoldDLP* ontologies took around half an hour with 1.000 individuals, and exceeded the predefined maximum time (three hours) with 10.000 individuals. The *Wine* ontology only processed 10 individuals in the maximum time;

- Inference is the most exigent procedure with all the ontologies. In the first three ontologies, inference was made with the maximum individuals in the predefined time. With the *Diseases*, *Social Network*, and *GoldDLP* ontologies, inference was only possible with 100 individuals. With more individuals, the available memory was exhausted. The *Wine* ontology only processed 10 individuals in the maximum time.

The bad inference results in the last four ontologies are mainly due to the fact that the *Wine* ontology contains *cardinality restrictions*, while the other three have *transitivity* properties. Both cardinality restrictions and transitivity generate first-order formulas with more than two free variables (see Appendix I). Since in Markov logic all the possible combinations between these free variables with the individuals of the domain must be grounded, the number of groundings increases exponentially with the number of free variables. This fact raises the complexity of reasoning, both in terms of time and used memory.

In the weight learning, the bad results are also derived from the same problem, because even if inference is not made (only discriminative learning uses inference), the number of groundings for each formula must be counted, and the more complex are the formulas, more intensive is the counting.
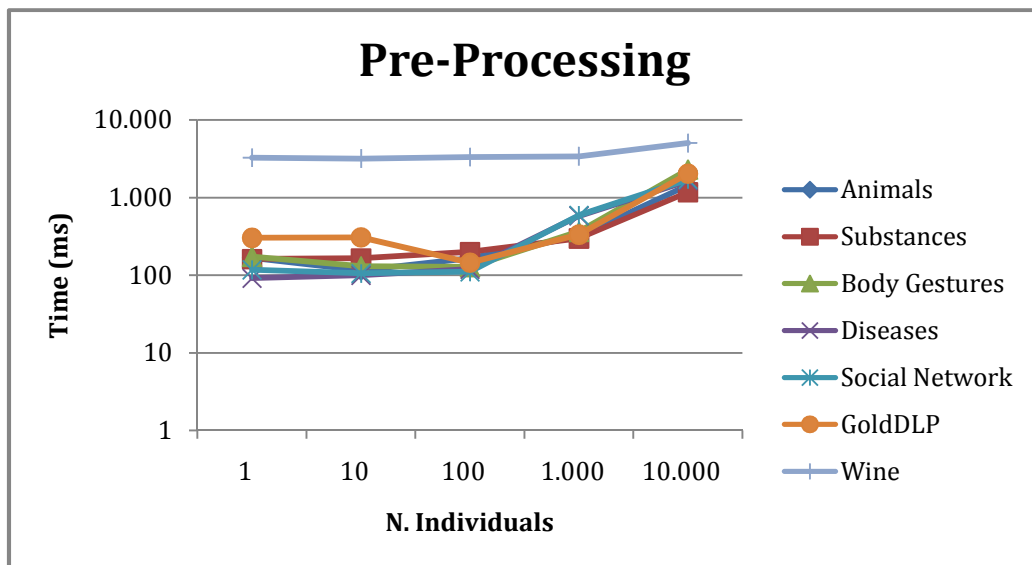

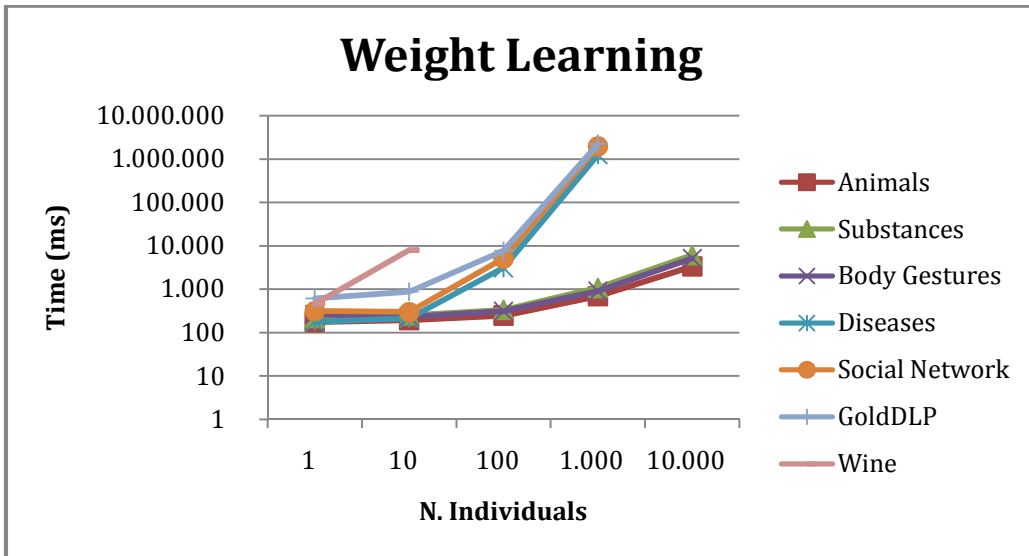**Figure 24.** Pre-Processing scalability results. Time is in logarithmic scale.

Figure 25. Markov logic weight learning scalability results. Time is in logarithmic scale.
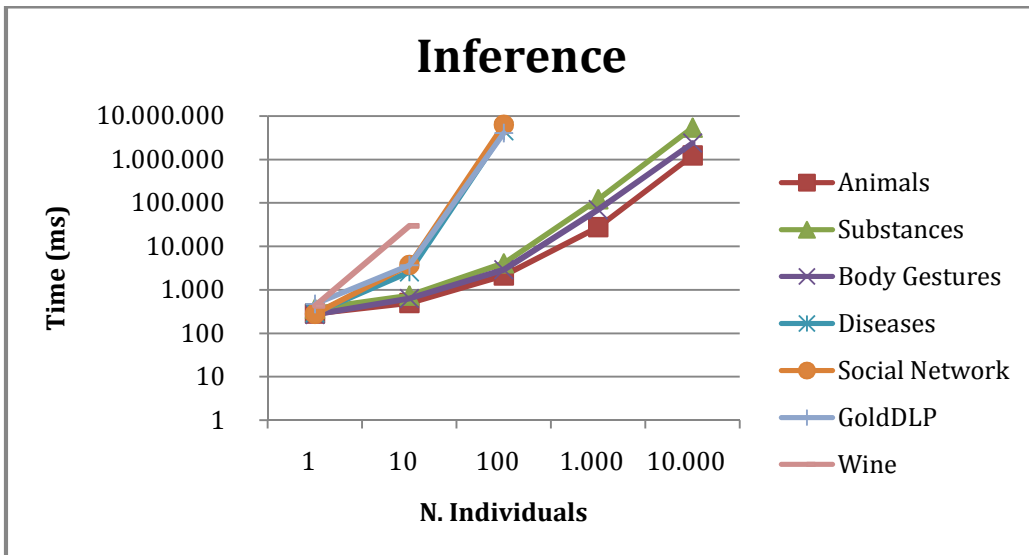

Figure 26. Markov logic inference scalability results. Time is in logarithmic scale.

5. Conclusions

To realize the Semantic Web vision of a world where knowledge is the most important element, mechanisms must be developed to represent and reason about the uncertainty that this knowledge could arise. In this thesis, we explored the use of Markov logic, a unifying representation of logic and probability, to learn and reason about uncertainty in the Semantic Web. The main contributions of this thesis are:

- Through Markov logic, we applied the ideas of a new field, statistical relational learning, which has been developing mechanisms to learn and reason in large uncertain relational domains. These characteristics are the same of the Semantic Web;

- Unlike other approaches, we studied not only reasoning about uncertainty in the Semantic Web, but also how we can learn that uncertainty from various sources, like ontology individuals, textual corpus, and web search engines;

- Unlike other approaches that also use probabilistic graphical models, our approach is based on Markov networks, which are undirected probabilistic models that allow cyclic knowledge;

- We developed a new method to transform probabilities in Markov logic weights. This method is more efficient than other approaches, and has no restrictions on his application;

- We developed a new method to learn the probabilities of OWL2 axioms using a web search engine. This method uses semantic similarity techniques, and can be applied in several domains where there is no other information about the uncertainty of the ontology;

- Our system, *Incerto*, can be used in many crucial domains for the Semantic Web, like ontology learning, social networks analysis, and ontology individual clustering.

In fact, since Markov logic is a so broad approach, we think that our approach of using Markov logic learning and reasoning capabilities in the Semantic Web can be seen as an introductory step in providing a general Markov logic framework for the Semantic Web (Pedro Domingos, Lowd, et al. 2008), providing services like ontology learning, reasoning, mapping, refining, among others.

However, currently, Markov logic contains some problems that interfere with its use in many Semantic Web domains. The high complexity on reasoning with transitivity and cardinality restrictions largely restricts its appliance in many ontologies. The lack of the definition of uncertainty information about evidential knowledge (i.e., we can only define the uncertainty of relations between classes and properties, and not about individuals belonging to a certain class or property) also reduces its usability in several domains, like ontology population and learning. To realize the vision of Markov logic as a general framework for the Semantic Web, these problems must be solved.

This thesis comprised several deliverables, concluded at different stages of the project. At the end of the project, the following milestones were delivered:

- *Bibliographic Revision*, describing the most relevant concepts to this thesis (Chapter 2);
- *Technologies and Tools Study*, describing the tools and technologies that could be used in this thesis (see Section 4.3);
- *Thesis Proposal Elaboration,* describing the proposed approach and the architecture of the system;
- *First System Prototype*, a simple version of the system, composed mainly by the System Core (see Section 4.3.1);
- *Second System Prototype*, the final version of the system, with all the features described in Chapter 4;
- *Final Thesis Elaboration*, describing all the work done in this thesis.

The work plan defined for this thesis, with tasks and respective schedule, is represented in Figure 27. Some deliverables suffered some delays compared to the schedule, but they were completely finished.

5.1. Future Work

In this section, we identify some directions for future work. Some of these directions have the objective of improving the executed work, while others explore some new interesting ideas and concepts that aroused during this work.

5.1.1. General Ideas

In general, there are several issues that deserve further exploration. The most obvious is the appliance of the presented techniques in more domains. We have applied our approaches in several relevant domains for the Semantic Web, like reasoning about automatically learned ontologies (Section 3.1.2) and web-based social networks (Section 3.2.1). More domains and tasks, like mapping and aligning ontologies (Euzenat and Shvaiko 2007), and automatically learning and augmenting ontologies (e.g., (Philipp Cimiano 2006), (Maedche 2002), (Suchanek, Sozio, and Weikum 2009)) could largely benefit from Markov logic capabilities.

In this thesis, we explored several ways to automatically learn the uncertainty of ontology axioms. However, more ideas could be explored:

- *Other ways of learning individuals.* We explored the use of textual resources to learn ontology individuals. Other way of populating ontologies is through the analysis of structured data, like relational databases or other ontologies. In this case, *mappings* (Euzenat and Shvaiko 2007) must be made between the structured data objects and the entities of the ontology.

- *Learn the uncertainties directly from textual corpus.* This is done by analyzing textual resources for patterns like "70% of A is B" or "Most of the A's are B's". This can be done by using previously trained classifiers or general lexico-syntactic rules.

- *Use the structure of the ontology.* The structure of the ontology can provide interesting information about the uncertainty of its axioms. Some other works (Gomes 2004) (Stuckenschmidt and Klein 2004) (Ramakrishnan et al. 2005) already explored similar approaches in ontologies, however with distinct objectives than ours. The field of *network analysis* (Brandes and Erlebach 2005) can provide us with some interesting concepts that can be potentially transferred to our specific case.

- *Collective learning of weights.* The idea is to learn the weights collectively from multiple ontologies about the same domain. This task can be achieved by exploring techniques from collective learning fields, like *relational reinforcement learning* (Tadepalli, Givan, and Driessens 2004).

- *Trust propagation.* The idea is to use the propagation of *trust metrics* in groups to automatically learn the uncertainty of certain axioms. This idea was already applied in the Markov logic context (M. Richardson 2004).

As seen in Section 0, there are some OWL2 properties, like *transitivity* and *cardinality restrictions*, which make reasoning in Markov logic very difficult in most of the domains. Some recent works (P. Singla and P. Domingos 2008) (Jain and Beetz 2008) are already exploring new techniques to cope with this kind of problems, and we plan to apply them in the future.

5.1.2. Probabilistic Reasoning in Uncertainty-annotated Ontologies

One of the properties of the proposed approach to interpret probabilities as weights in Markov logic that needs further study is the influence of the number of individuals used in the learning process. During our experimentation, we found that in certain domains the number of individuals used in the learning process largely influences the learned weights. These weights, sometimes, did not reflect the desired probabilities, giving very polarized probabilities during inference. It is studied (Jain, Kirchlechner, and Beetz 2007) that the number of individuals can influence the learned weights in Markov logic, and we plan to study this situation in more depth in the future. We also plan to study the difficulties of convergence of the discriminative learning algorithm in some domains, fact that influences the quality of the learned weights.

5.1.3. Probabilistic Reasoning by Learning Individuals/Probabilities

Learn Individuals

Several improvements can be made to the ontology population process. Improving the linguistic processing of the corpus, for example by adding support to others languages besides English and adding stemmers to help lemmatizers, are some of these improvements. Other tasks related to the mapping of the learned individuals to the classes and individuals already existent in the source ontologies, like *coreference resolution* (P. Singla and P. Domingos 2006b) (Culotta, Wick, Hall, and McCallum 2007), *word-sense disambiguation* (Navigli 2009), and *canonicalization* (Culotta, Wick, Hall, Marzilli, et al. 2007), could also improve the quality of the results. One interesting idea, as proposed by (Suchanek, Sozio, and Weikum 2009), is to use the ontology structure and the already available individuals to guide the learning process. Other improvements, like learning new lexico-syntactic patterns and develop specific extractors to parse lists (Etzioni et al. 2005), could also improve the number of extracted individuals.

Learn Probabilities

In this thesis, we explored 6 semantic similarity metrics to learn axioms probabilities using web search engines. In the future, more metrics (e.g., (Lin 1998) (Bollegala, Matsuo, and Ishizuka 2007)) could be also explored. More complex ways of normalizing metrics results, like the ones that use *Naïve Bayes Classifiers* (Etzioni et al. 2005), even if more complex, usually give better results, and could be an issue for further exploration.

5.1.4. System

One of the most interesting future tasks is to develop a domain-specific Java Markov logic reasoning engine that could be easily incorporated in the existing system. This engine would be optimized and specifically developed for the Semantic Web domain, providing only the algorithms that perform well and are needed in this domain. This engine would also solve one of the major bottlenecks of the system, which is the need to communicate with external processes, since the available Markov logic engines were not developed in Java.

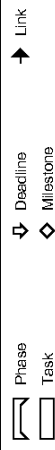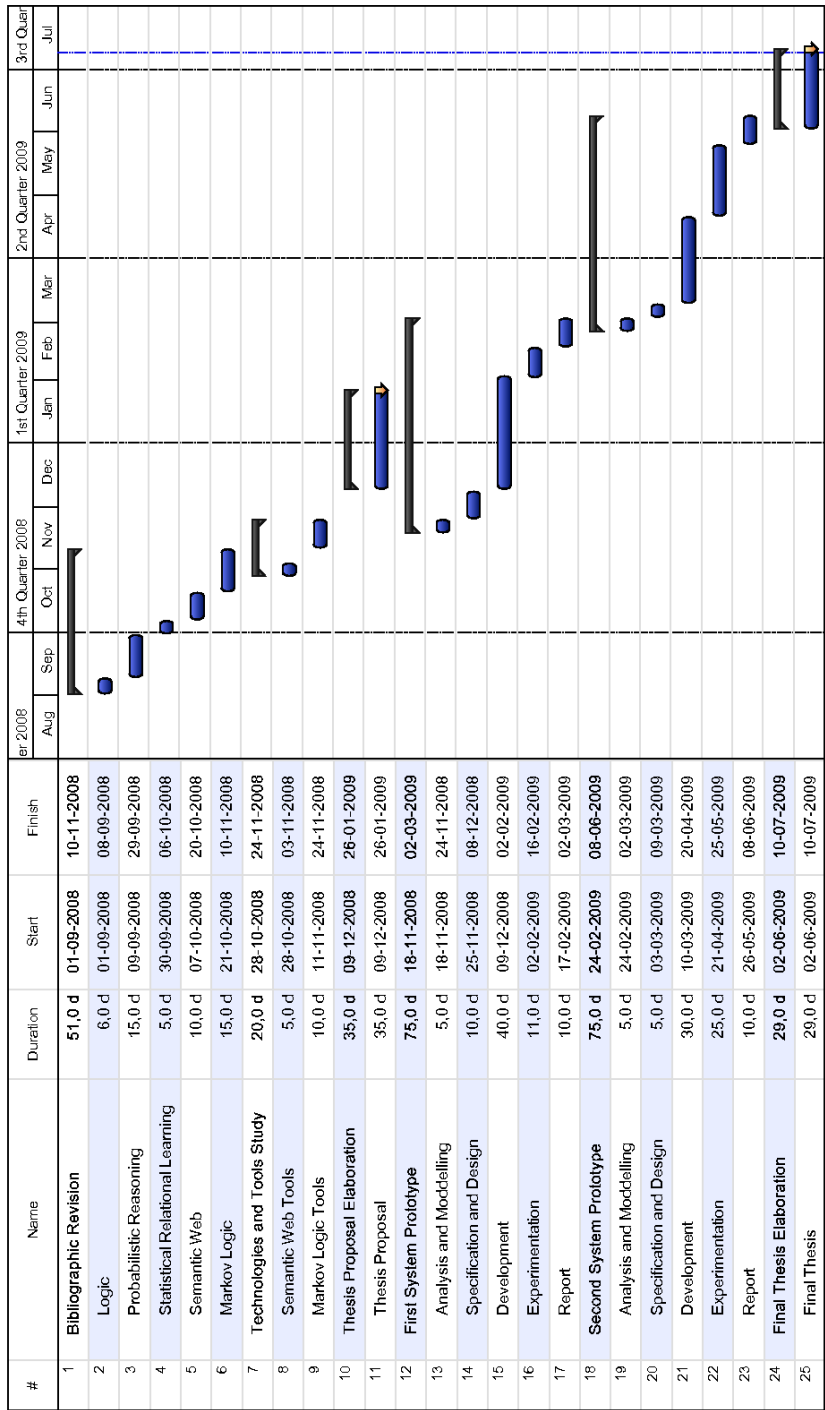| #	Name	Duration	Start	Finish
1	**Bibliographic Revision**	**51,0 d**	**01-09-2008**	**10-11-2008**
2	Logic	6,0 d	01-09-2008	08-09-2008
3	Probabilistic Reasoning	15,0 d	09-09-2008	29-09-2008
4	Statistical Relational Learning	5,0 d	30-09-2008	06-10-2008
5	Semantic Web	10,0 d	07-10-2008	20-10-2008
6	Markov Logic	15,0 d	21-10-2008	10-11-2008
7	Technologies and Tools Study	20,0 d	28-10-2008	24-11-2008
8	Semantic Web Tools	5,0 d	28-10-2008	03-11-2008
9	Markov Logic Tools	10,0 d	11-11-2008	24-11-2008
10	Thesis Proposal Elaboration	35,0 d	09-12-2008	26-01-2009
11	Thesis Proposal	35,0 d	09-12-2008	26-01-2009
12	First System Prototype	75,0 d	18-11-2008	02-03-2009
13	Analysis and Moddelling	5,0 d	18-11-2008	24-11-2008
14	Specification and Design	10,0 d	25-11-2008	08-12-2008
15	Development	40,0 d	09-12-2008	02-02-2009
16	Experimentation	11,0 d	02-02-2009	16-02-2009
17	Report	10,0 d	17-02-2009	02-03-2009
18	Second System Prototype	75,0 d	24-02-2009	08-06-2009
19	Analysis and Moddelling	5,0 d	24-02-2009	02-03-2009
20	Specification and Design	5,0 d	03-03-2009	09-03-2009
21	Development	30,0 d	10-03-2009	20-04-2009
22	Experimentation	25,0 d	21-04-2009	25-05-2009
23	Report	10,0 d	26-05-2009	08-06-2009
24	Final Thesis Elaboration	29,0 d	02-06-2009	10-07-2009
25	Final Thesis	29,0 d	02-06-2009	10-07-2009

Figure 27. Project planning.

References

Antoniou, Grigoris, and Frank van Harmelen. 2008. *A Semantic Web Primer, 2nd Edition*. 2nd ed. The MIT Press.

Baader, Franz, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. 2007. *The Description Logic Handbook: Theory, Implementation, and Applications*. 2nd ed. Cambridge University Press.

Baeza-Yates, Ricardo, and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. 1st ed. Addison Wesley.

Barwise, Jon, and John Etchemendy. 2002. *Language, Proof and Logic*. Center for the Study of Language and Inf.

Bar-Yossef, Ziv, and Maxim Gurevich. 2008. Random sampling from a search engine's index. *Journal of the ACM* 55, no. 5: 1-74.

Berners-Lee , Tim. 2005. RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. January. http://tools.ietf.org/html/rfc3986.

Berners-Lee, T., J. Hendler, and O. Lassila. 2001. The Semantic Web. *Scientific American* 284, no. 5: 28-37.

Berners-Lee, Tim. 1998. Semantic Web roadmap. http://www.w3.org/DesignIssues/Semantic.html.

---. 2000. Semantic Web. In *Invited Talk at XML 2000 Conference*. http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html.

Besag, J. 1975. Statistical analysis of non-lattice data. *The Statistician* 24, no. 3: 179-195.

Bishop, Christopher M. 2007. *Pattern Recognition and Machine Learning*. 1st ed. Springer.

Bobillo, F., and U. Straccia. 2008. fuzzyDL: An expressive fuzzy description logic reasoner. In *Proceeding of the IEEE International Conference on Fuzzy Systems*, 923-930.

Bollegala, D, Y Matsuo, and M Ishizuka. 2007. Measuring semantic similarity between words using web search engines. In *Proceedings of the 16th international conference on World Wide Web*, 757-766. Banff, Alberta, Canada: ACM. http://portal.acm.org/citation.cfm?id=1242675&dl=GUIDE,

Brandes, Ulrik, and Thomas Erlebach. 2005. *Network Analysis: Methodological Foundations*. 1st ed. Springer, March 24.

Breitman, K.K., M.A. Casanova, and W. Truszkowski. 2006. *Semantic Web: Concepts, Technologies and Applications*. 1st ed. Springer.

Cardoso, Jorge. 2007. *Semantic Web Services: Theory, Tools and Applications*. IGI Global.

Church, K. W., and P. Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational linguistics* 16, no. 1: 22-29.

Cilibrasi, R., and P. M. B. Vitanyi. 2004. The google similarity distance. *Arxiv preprint cs/0412098*.

Cimiano, P., G. Ladwig, and S. Staab. 2005. Gimme' The Context: Context driven Automatic Semantic Annotation with CPANKOW. In *Proceedings of the WWW*.

Cimiano, Philipp. 2006. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. 1st ed. Springer.

Costa, P. C. G., and K. J. Laskey. 2005. PR-OWL: A Bayesian Ontology Language for the Semantic Web. In *Proceedings of the 1st Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2005)*, 6-10.

Culotta, A., M. Wick, R. Hall, M. Marzilli, and A. McCallum. 2007. Canonicalization of database records using adaptive similarity measures. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 201-209. ACM New York, NY, USA.

Culotta, A., M. Wick, R. Hall, and A. McCallum. 2007. First-order probabilistic models for coreference resolution. In *Proceedings of NAACL HLT*, 81-88.

Cunningham, D. H., D. D. Maynard, D. K. Bontcheva, and M. V. Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)* (July).

Ding, Zhongli, Yun Peng, and Rong Pan. 2006. BayesOWL: Uncertainty Modeling in Semantic Web Ontologies. In *Soft Computing in Ontologies and Semantic Web*, 3-29.

Dolby, J., A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, and L. Ma. 2007. Scalable Semantic Retrieval through Summarization and Refinement. In *Proceedings of the National Conference on Artificial Intelligence*, 22:299. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press.

Domingos, Pedro, Stanley Kok, Daniel Lowd, Hoifung Poon, Matthew Richardson, and Parag Singla. 2008. Markov Logic. In *Probabilistic Inductive Logic Programming*, 92-117.

Domingos, Pedro, Daniel Lowd, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. 2008. Just Add Weights: Markov Logic for the Semantic Web. In *Uncertainty Reasoning for the Semantic Web I*, 1-25.

Etzioni, O., M. Cafarella, D. Downey, A. M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence* 165, no. 1: 91-134.

Euzenat, Jérôme, and Pavel Shvaiko. 2007. *Ontology Matching*. 1st ed. Springer.

Evans, R., and S. Street. 2004. A framework for named entity recognition in the open domain. In *Proceedings of Recent Advances in Natural Language Processing (RANLP-2003)*, 137-144. Borovetz, Bulgaria, September.

Fellbaum, Christiane. 1998. *WordNet: An Electronic Lexical Database*. The MIT Press.

Fukushige, Y. 2005. Representing Probabilistic Relations in RDF. In *Proceedings of Workshop on Uncertainty Reasoning for the Semantic Web (URSW)*.

Getoor, Lise. 2003. Link mining: a new data mining challenge. *SIGKDD Explor. Newsl.* 5, no. 1: 84-89.

Getoor, Lise, and Christopher P. Diehl. 2005. Link mining: a survey. *SIGKDD Explor. Newsl.* 7, no. 2: 3-12.

Getoor, Lise, and Ben Taskar. 2007. *Introduction to Statistical Relational Learning*. The MIT Press.

Getoor, U., N. Friedman, D. Roller, A. Pfeffer, and B. Taskar. 2007. Probabilistic Relational Models. In *Introduction to Statistical Relational Learning*, 129-174. The MIT Press.

Giugno, Rosalba, and Thomas Lukasiewicz. 2002. P-SHOQ(D): A Probabilistic Extension of SHOQ(D) for Probabilistic Ontologies in the Semantic Web. In *Proceedings of the European Conference on Logics in Artificial Intelligence*, 86-97. Springer-Verlag.

Gomes, Paulo. 2004. Software Design Retrieval Using Bayesian Networks and WordNet. In *Advances in Case-Based Reasoning*, 184-197.

Grau, B. C., I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. 2008. OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*.

Gu, T., H. K. Pung, D. Zhang, and G. Kotsis. 2004. A Bayesian approach for dealing with uncertain contexts. In *The Proceeding of the Second International Conference on Pervasive Computing*.

Haarslev, V., and R. Möller. 2003. Racer: A Core Inference Engine for the Semantic Web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools*, 27-36.

Hearst, M. A. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, 539-545. Association for Computational Linguistics Morristown, NJ, USA.

Heckerman, D., C. Meek, and D. Koller. 2007. Probabilistic Entity-Relationship Models, PRMs, and Plate Models. In *Introduction to Statistical Relational Learning*, 201-238. The MIT Press.

Hendler, James. 2001. Agents and the Semantic Web. *IEEE Intelligent Systems* 16, no. 2: 30-37.

Henrik, Nottelmann , and Fuhr   Norbert. 2006. Adding Probabilities and Rules to Owl Lite Subsets Based on Probabilistic Datalog. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 14, no. 1: 17-42.

Holi, Markus, and Eero Hyvönen. 2006. Modeling Uncertainty in Semantic Web Taxonomies. In *Soft Computing in Ontologies and Semantic Web*, 31-46.

Horridge, M., S. Bechhofer, and O. Noppens. 2007. Igniting the OWL 1.1 Touch Paper: The OWL API. In *Proc. of the Third International OWL:Experiences and Directions Workshop (OWLED- 2007)*. Innsbruck, Austria .

Jain, Dominik, and Michael Beetz. 2008. Towards Accurate Models of Joint Probability Distributions in Relational Domains: Cardinality Constraints and Dynamic Parameters. Technical Report. Technische Universität München.

Jain, Dominik, Bernhard Kirchlechner, and Michael Beetz. 2007. Extending Markov Logic to Model Probability Distributions in Relational Domains. In *KI 2007: Advances in Artificial Intelligence*, 129-143.

Jordan, Michael I. 2004. Graphical Models. *Statistical Science*. Special Issue on Bayesian Statistics.

Jurafsky, Daniel, and James H. Martin. 2008. *Speech and Language Processing (2nd Edition)*. 2nd ed. Prentice Hall.

Kersten, M., and G. C. Murphy. 2006. Using task context to improve programmer productivity. In *Proceedings of the 13th ACM SIGSOFT 14th International Symposium on Foundations of Software Engineering*, 1-11. ACM Press New York, NY, USA.

Klinov, Pavel. 2008. Pronto: A Non-monotonic Probabilistic Description Logic Reasoner. In *The Semantic Web: Research and Applications*, 822-826.

Klinov, Pavel, and Bijan Parsia. 2008. Probabilistic Modeling and OWL: A User Oriented Introduction to P-SHIQ(D). In *Proc. of the Fifth OWL: Experiences and Directions Workshop 2008 (OWLED'08)*. October 26.

Klir, George J., and Bo Yuan. 1995. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. 1st ed. Prentice Hall PTR.

Kok, S., and P. Domingos. 2005. Learning the structure of Markov logic networks. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, 22:441-448.

Kok, S., P. Singla, M. Richardson, and P. Domingos. 2007. *The Alchemy system for statistical relational AI*. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. http://alchemy.cs.washington.edu.

de Kunder, Maurice. 2009. WorldWideWebSize.com | The size of the World Wide Web. http://www.worldwidewebsize.com/.

Laskey, K. B., and P. C. G. Costa. 2005. Of Klingons and Starships: Bayesian Logic for the 23rd Century. In *Proceedings of the Twenty-first Conference on Uncertainty in Artificial Intelligence* . AUAI Press.

Levesque, H. J., and R. J. Brachman. 1985. A fundamental tradeoff in knowledge representation and reasoning. *Readings in Knowledge Representation* 1.

Lin, D. 1998. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 296-304.

Lindley, Dennis V. 2006. *Understanding Uncertainty*. 1st ed. Wiley-Interscience.

Lukasiewicz, T. 2008. Expressive probabilistic description logics. *Artificial Intelligence* 172, no. 6-7: 852-883.

Lukasiewicz, Thomas, and Umberto Straccia. 2008. Managing Uncertainty and Vagueness in Description Logics for the Semantic Web. *Web Semantics Sci Serv Agents World Wide Web*.

Maedche, Alexander. 2002. *Ontology Learning for the Semantic Web*. 1st ed. Springer.

Magnini, B., M. Negri, R. Prevete, and H. Tanev. 2002. Is it the right answer? Exploiting web redundancy for answer validation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 425–432.

Marques de Sá, J.P. . 2001. *Pattern Recognition: Concepts, Methods and Applications*. 1st ed. Springer.

McDowell, L. K., and M. Cafarella. 2008. Ontology-driven, unsupervised instance population. *Web Semantics: Science, Services and Agents on the World Wide Web* 6, no. 3: 218-236.

Mihalkova, L., and R. J. Mooney. 2007. Bottom-up learning of Markov logic network structure. In *Proceedings of the 24th International Conference on Machine Learning*, 625-632. ACM Press New York, NY, USA.

Mika, Peter. 2007. *Social Networks and the Semantic Web*. 1st ed. Springer.

Milch, B., B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Rolobov. 2007. BLOG: Probabilistic Models with Unknown Objects. In *Introduction to Statistical Relational Learning*, 373-398. The MIT Press.

Milch, B., and S. Russell. 2006. General-purpose MCMC inference over relational structures. In *Proc. 22nd Conference on Uncertainty in Artificial Intelligence*, 349–358.

Milch, B., L. S. Zettlemoyer, K. Kersting, M. Haimes, and L. P. Kaelbling. 2008. Lifted Probabilistic Inference with Counting Formulas. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 1062-1068.

Moller, Ralf, and Volker Haarslev. 2008. Tableau-based Reasoning. In *Handbook of Ontologies*. 2nd ed. Springer.

Motik, Boris, and Ulrike Sattler. 2006. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In *Logic for Programming, Artificial Intelligence, and Reasoning*, 227-241.

Motik, Boris, Rob Shearer, and Ian Horrocks. 2007. Optimized Reasoning in Description Logics Using Hypertableaux. In *Automated Deduction – CADE-21*, 67-83.

Muggleton, S., and N. Pahlavi. 2007. Stochastic Logic Programs: A Tutorial. In *Introduction to Statistical Relational Learning*, 323-338. The MIT Press.

Nath, T. H., and R. Moller. 2008. ContraBovemRufum: A System for Probabilistic Lexicographic Entailment. *Proceedings of the 21st International Workshop on Description Logics (DL2008)*.

Navigli, Roberto. 2009. Word sense disambiguation: A survey. *ACM Comput. Surv.* 41, no. 2: 1-69.

Neville, J., and D. Jensen. 2007. Relational Dependency Networks. *The Journal of Machine Learning Research* 8: 653-692.

Nottelmann, Henrik, and Umberto Straccia. 2006. A probabilistic, logic-based framework for automated Web directory alignment. *Soft computing in ontologies and the semantic Web. Studies in fuzziness and soft computing*: 47--77.

Pan, J. Z., G. Stamou, G. Stoilos, and E. Thomas. 2007. Expressive Querying over Fuzzy DL-Lite Ontologies. In *Proceedings of the International Workshop on Description Logics (DL 2007)*.

Pan, Rong, Zhongli Ding, Yang Yu, and Yun Peng. 2005. A Bayesian Network Approach to Ontology Mapping. *Proceedings of the International Semantic Web Conference 2005*: 563--577.

Pearl, Judea. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. 1st ed. Morgan Kaufmann.

Picard, R. W. 1997. *Affective computing*. MIT press.

Poon, H., and P. Domingos. 2006. Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. In *Proceedings of the National Conference on Artificial Intelligence*, 21:458-463. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Predoiu, Livia, and Heiner Stuckenschmidt. 2007. A probabilistic Framework for Information Integration and Retrieval on the Semantic Web. In *Proc. of 3rd International Workshop on Database Interoperability (InterDB) in conjunction with the VLDB conference*, 23-28. Vienna, Austria.

Ramakrishnan, Cartic, William H. Milnor, Matthew Perry, and Amit P. Sheth. 2005. Discovering informative connection subgraphs in multi-relational graphs. *SIGKDD Explor. Newsl.* 7, no. 2: 56-63.

Rehman Abbasi, Abdul, Nitin V. Afzulpurkar, and Takeaki Uno. 2008. Exploring Un-Intentional Body Gestures for Affective System Design . In *Affective Computing*. Jimmy Or. InTech Education and Publishing , May.

Rersting, R., and L. De Raedt. 2007. Bayesian Logic Programming: Theory and Tool. In *Introduction to Statistical Relational Learning*, 291-322. The MIT Press.

Riazanov, A. 2002. The design and implementation of VAMPIRE. *AI Communications* 15, no. 2: 91-110.

Richardson, M. 2004. Learning and Inference in Collective Knowledge Bases. PhD Thesis, University of Washington.

Richardson, M., and P. Domingos. 2006. Markov logic networks. *Machine Learning* 62, no. 1: 107-136.

Richardson, Matthew, Rakesh Agrawal, and Pedro Domingos. 2003. Trust Management for the Semantic Web. In *The SemanticWeb - ISWC 2003*, 351-368.

Roller, D., N. Friedman, L. Getoor, and B. Taskar. 2007. Graphical Models in a Nutshell. In *Introduction to Statistical Relational Learning*, 13-55. The MIT Press.

Russell, Stuart, and Peter Norvig. 2002. *Artificial Intelligence: A Modern Approach (2nd Edition)*. 2nd ed. Prentice Hall.

Sanchez, Elie. 2006. *Fuzzy Logic and the Semantic Web*. 1st ed. Elsevier Science.

Sauermann, Leo , and Richard Cyganiak. 2008. Cool URIs for the Semantic Web. March 31. http://www.w3.org/TR/cooluris/.

Selman, B., H. A. Kautz, and B. Cohen. 1993. Local search strategies for satisfiability testing. *Proceedings of the Second DIMACS Challenge on Cliques, Coloring, and Satisfiability*.

Shapiro, S. 2001. Classical logic II–Higher-order logic. *The Blackwell Guide to Philosophical Logic*.

Silva, Paulo Pinheiro da, Deborah L. McGuinness, and Richard Fikes. 2006. A proof markup language for Semantic Web services. *Information Systems* 31, no. 4-5 (July): 381-395.

Singla, P., and P. Domingos. 2005. Discriminative Training of Markov Logic Networks. In *Proceedings of the National Conference on Artificial Intelligence*, 20:868. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

---. 2006a. Memory-Efficient Inference in Relational Domains. In *Proceedings of the National Conference on Artificial Intelligence*, 21:488. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

---. 2006b. Entity resolution with markov logic. In *Proceedings of the Sixth IEEE International Conference on Data Mining*, 572–582.

---. 2008. Lifted first-order belief propagation. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*. Chicago, IL: AAAI Press.

Sirin, E., B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. 2007. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web* 5, no. 2: 51-53.

Sterling, Leon, and Ehud Shapiro. 1994. *The Art of Prolog, Second Edition: Advanced Programming Techniques*. 2nd ed. The MIT Press.

Stocker, Markus, and Michael Smith. 2008. Owlgres: A Scalable OWL Reasoner. In *Proceedings of the Fifth International Workshop OWL: Experiences and Directions (OWLED 2008)* . Karlsruhe, Germany.

Stoilos, G., G. Stamou, V. Tzouvaras, J. Z. Pan, and I. Horrocks. 2005a. Fuzzy OWL: Uncertainty and the Semantic Web. In *Proceedings of the International Workshop on OWL: Experiences and Directions*.

---. 2005b. The fuzzy description logic f-SHIN. In *Proc. of the International Workshop on Uncertainty Reasoning for the Semantic Web*, 67–76.

Straccia, U. 2006a. A Fuzzy Description Logic for the Semantic Web. *Fuzzy Logic and the Semantic Web*: 73-90.

---. 2006b. Answering vague queries in fuzzy DL-Lite. In *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-06)*, 2238–2245.

Stuckenschmidt, Heiner, and Michel Klein. 2004. Structure-Based Partitioning of Large Concept Hierarchies. In *The Semantic Web – ISWC 2004*, 289-303.

Suchanek, Fabian M, Mauro Sozio, and Gerhard Weikum. 2009. SOFIE: A Self-Organizing Framework for Information Extraction. In . Madrid, Spain, April.

Tadepalli, P., R. Givan, and K. Driessens. 2004. Relational Reinforcement Learning: An Overview. In *Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning*, 4:1–9.

Tanev, H., and B. Magnini. 2006. Weakly supervised approaches for ontology population. *Proceedings of EACL-2006, Trento*: 3-7.

Taskar, B., P. Abbeel, M. F. Wong, and D. Roller. 2007. Relational Markov Networks. In *Introduction to Statistical Relational Learning*, 175-200. The MIT Press.

The Unicode Consortium. 2006. *Unicode Standard, Version 5.0, The*. 5th ed. Addison-Wesley Professional.

Tsarkov, Dmitry, and Ian Horrocks. 2006. FaCT++ Description Logic Reasoner: System Description. In *Automated Reasoning*, 292-297.

Tsarkov, Dmitry, Alexandre Riazanov, Sean Bechhofer, and Ian Horrocks. 2004. Using Vampire to Reason with OWL. In *The Semantic Web – ISWC 2004*, 471-485.

Tversky, Amos, and Daniel Kahneman. 1974. Judgment under Uncertainty: Heuristics and Biases. *Science* 185, no. 4157: 1124-1131.

Udrea, O., V. S. Subrahmanian, and Z. Majkic. 2006. Probabilistic RDF. In *IEEE International Conference on Information Reuse and Integration*, 172-177.

Van Dongen, Stijn. 2000. Graph Clustering by Flow Simulation. PhD Thesis, University of Utrecht, May.

Wahlster, Wolfgang, Henry Lieberman, and James Hendler. 2002. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. The MIT Press.

Wei, W., J. Erenrich, and B. Selman. 2004. Towards Efficient Sampling: Exploiting Random Walk Strategies. In *Proceedings of the National Conference on Artificial Intelligence*, 670-676. San Jose, CA: AAAI Press.

Yang, Y., and J. Calmet. 2005. OntoBayes: An Ontology-Driven Uncertainty Model. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, 1:457-463. IEEE Computer Society Washington, DC, USA.

Zhao, Y., and G. Karypis. 2005. *Criterion functions for document clustering*. University of Minnesota.

Appendix I

OWL2 interpretation as first-order logic.