UNIVERSITÀ DI PISA
DIPARTIMENTO DI INFORMTICA
Master Degree in Computer Science

Learning Fuzzy $\mathcal{EL}(D)$ Inclusion Axioms from Crisp OWL Ontologies

Author:
MUCCI Matteo
Matricola:480535

Supervisor:
STRACCIA Umberto

Examiner:
MILAZZO Paolo

Academic Year: 2013-2014

Contents

Abstract

This work aims at defining, implementing and comparing some algorithms that automatically learn specific OWL 2 inclusion axioms. These axioms describe sufficient conditions of a given target OWL 2 concept. To do so, a crisp OWL 2 background ontology, a target concept and a crisp training set are given. The specific learnt inclusion axioms belong to the fuzzy OWL 2 EL Profile language. Fuzziness has been added in order to improve the readability of the induced axioms. For instance, we may learn that 'a good hotel is one that has a *low price*', in place of 'a good hotel is one that whose price is between 30 and 60 euro'. Here the fuzzy concept 'low price' has been automatically determined from the training data.

The algorithms taken we have worked out are FOIL, a probabilistic variant of FOIL (pFOIL), a genetic variant of FOIL (gFOIL), the combination of the latter two (pgFOIL) and an AdaBoost variant of gFOIL (gAdaBoost).

FOIL learns one axiom trying to greedily maximizing a score function. After having learnt one axiom the positives samples covered are removed. The procedure is iterated until a given coverage threshold is reached.

pFOIL tries to learn axioms by taking into account the ensemble of learnt axioms, and specifically evaluates the ensemble in probabilistic terms.

gFOIL exploits hybrid learning to learn one axiom. Hybrid learning is a particular form of genetic programming that we adapted to cope with an ontology background theory. Like FOIL, the algorithm evaluates one axiom at time and removes the positive samples covered.

pgFOIL extends gFOIL by evaluating the score of a learnt axiom like for pFOIL.

gAdaBoost is based on a modified version of Real AdaBoost, in which the weak learner used is as for hybrid learning.

Finally, a validation procedure has been adopted to evaluate the implemented learning algorithms.

Introduction

The study and usage of Descriptive Logics (DLs) to manage knowledge bases is nowadays wide spread and are at the core of the *OWL 2* [5, 32] standard for the definition of ontologies (knowledge bases) and numerous ontologies exists nowadays [1].

However, so far, the application of machine learning methods in the context of *OWL 2* is relatively rather an unexplored area, and, specifically, related to the induction of rules describing a target *OWL 2* concept [27].

This work aims at defining, implementing and comparing several algorithms to induce such rules. Some of the of these algorithms are inspired by *Inductive Logic Programming* (ILP)[9], a widely explored research area, and our aim is to see if and how these algorithms can be adapted to the *OWL 2* context.

Beside the definition of such algorithms, we address another issue. Many inductive algorithms can reach a good effectiveness, but can also produce rules that are hard to be understood for a human being. The readability, besides effectiveness, is also important. In order to achieve better human readability of the induced rules, we use fuzziness to describe concepts. So, for instance, if we are dealing with an ontology about Hotels and we are interest in getting a description of what characterises good hotels, based on the data at hand, instead of learning restriction on prices like one that says a hotel has a price lower than 50 (so shall I consider a hotel whose price is above 50 as not good?), we may lear rather that a good hotel is one that has a *low price*. This latter concept is called a *fuzzy set* [38]: fuzzy sets have characteristic functions that are functions mapping elements into $[0, 1]$ instead of $\{0, 1\}$. Therefore, elements of a domain belong to a fuzzy set to some degree in $[0, 1]$ and, thus, in our example, the degree of goodness of a hotel is a degree in $[0, 1]$ that may depend on the hotel's price. The use of fuzzy concepts has been shown to improve the readability of logical axioms. So, our goal here is then to find a set of rules expressed as fuzzy *OWL 2* inclusion axioms.

[1] See, *e.g.*, `http://owl.cs.manchester.ac.uk/tools/repositories/`.

In this thesis, we start with FOIL ([27]), a method that was already used to induce fuzzy OWL axioms. From FOIL we adapt nFOIL ([22]) to our context and define a new algorithm, called pFOIL. Then, we try to use hybrid learning ([23]) in combination with FOIL and pFOIL to evaluate the change in effectiveness. Finally, we adapt a variant of AdaBoost [11], called $\mathbb{R}$eal AdaBoost [30], to our needs and obtained an algorithm that uses hybrid learning as weak learner. All these algorithms will be then tested for their effectiveness.

Part I

Knowledge Representation

Chapter 1

Web Ontology Language OWL

Among the various fields of interest of Artificial Intelligence (AI), there is the area of machine learning from knowledge bases. This work aims at finding ways to induce rules (*automatic learning*) that can give an understandable description of a certain concept expressed in terms of an underlying *knowledge representation* language.

Web Ontology Language (OWL) is a formalism to represent knowledge. It is based on DLs and can easily be transformed into natural language expressions. This formalism has been firstly defined as a **W3C** standard in 2004 ([31]). Then in 2009 it has been updated into *OWL 2* ([32]).

Both for *OWL* and *OWL 2* some profiles, have been defined [33]. An *OWL 2* profile is a sublanguage that limits expressive power but also decrease the complexity of reasoning. So beside a good understandability *OWL* and *OWL 2* give the user the possibility to balance complexity and expressiveness in accordance with his needs.

1.1 Description Logics Basics

Description Logics (DLs [7, 20]) can be considered the theoretical counterpart of *OWL* and *OWL 2* languages as *OWL* language constructs can be mapped into DLs, for which reasoning algorithms and computational complexity are known, allowing, thus, to reason with *OWL*, *OWL 2* and its profile.

For what concerns us:

- *OWL 2* refers to the DL $\mathcal{SROIQ}$ [5, 19]
- *OWL 2* EL refers to the DL $\mathcal{EL}$ family, more specifically to $\mathcal{EL}^{++}$ [1, 3, 13]

DLs have three fundamental blocks *concepts*, *roles* and *individuals*. Concepts can be seen as unary predicates or class, roles can be considered binary relations and individuals as elements that can belong to a concept. Complex concepts and roles can be built using different concept and role constructors.

1.1.1 DL families

DLs can be grouped into families, and each family consists of various logics each of which has a different expressive power and complexity. Families are represented by letters and each letter adds one or more constructors that allow to build complex concepts and roles. The basic family is the $\mathcal{AL}$ family that contains essential constructors. By adding "letters" to $\mathcal{AL}$ we can obtain more expressive expressions. However, the more expressive a logic the higher its computational complexity of the related reasoning algorithms.

1.1.2 $\mathcal{AL}$ family

$\mathcal{AL}$ (*Attributive Language*) family has few basic constructors. Considering A an atomic concept, C, D complex concepts and R an atomic role we have that a concept in $\mathcal{AL}$ is defined as ([37]):

C, D	$\rightarrow$	A	$\mid$	(atomic concept)
		$\top$	$\mid$	(universal concept)
		$\bot$	$\mid$	(bottom concept)
		$\neg A$	$\mid$	(atomic negation)
		$C \sqcap D$	$\mid$	(concept conjunction)
		$\forall R.C$	$\mid$	(universal qualified restriction)
		$\exists R.\top$		(existential unqualified restriction)

It is important to note that negation can only be applied to atomic concepts.

An informal *First Order* semantic can be considered as:

Syntax				FOL View
C, D	$\rightarrow$	A	$\mid$	$A(x)$
		$\top$	$\mid$	$\top(x)$
		$\bot$	$\mid$	$\bot\ (x)$
		$\neg A$	$\mid$	$\neg A(x)$
		$C \sqcap D$	$\mid$	$C(x) \wedge D(x)$
		$\forall R.C$	$\mid$	$\forall y.R(x, y) \rightarrow C(x)$
		$\exists R.\top$		$\exists y.R(x, y)$

Then an ontology can be defined as a pair $\langle \mathcal{A}, \mathcal{T} \rangle$, where $\mathcal{A}$ is called *ABox* and consists of a finite set of *concepts and roles assertion axioms*, while T is called *TBox* and is a finite set of *General Inclusion Axioms* (GCIs).

Each *ABox* axiom is of the form $a : C$ and $(a, b) : R$. They state respectively that *a is an instance of C* and *a and b are related through R* or *a has b as R*.

Each *TBox* axiom is of the form $C \sqsubseteq D$ (*General Concept Inclusion, or GCI*) or $C = D$ (*definitional*), where C and D are concepts. A GCI states that each instance of C is an instance of D or in FOL $\forall x.C(x) \rightarrow D(x)$ while a definition inclusion axiom is equivalent to say that $C \sqsubseteq D$ and $D \sqsubseteq C$, *i.e.*, in FOL $\forall x.C(x) \leftrightarrow D(x)$.

We can define an interpretation $\mathcal{I}$ as a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ where $\Delta^{\mathcal{I}}$ is non empty set called *interpretation domain* and $\cdot^{\mathcal{I}}$ is called *interpretation function* and maps a *concept* into a subset of $\Delta^{\mathcal{I}}$, a *role* into a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and an *individual* into an element of $\Delta^{\mathcal{I}}$.

The interpretation function is extended to complex concepts as follows:

$$\begin{array}{rcc} \top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \ , \\ \bot^{\mathcal{I}} & = & \emptyset \ , \\ (C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \ , \\ (\forall R.C)^{\mathcal{I}} & = & \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x) \subseteq C^{\mathcal{I}}\} \ , \\ (\exists R.\top)^{\mathcal{I}} & = & \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x) \neq \emptyset\} \ , \end{array}$$

where $R^{\mathcal{I}}(x) = \{y \in \Delta^{\mathcal{I}} \mid \langle x, y \rangle \in R^{\mathcal{I}}\}$.

Furthermore, we say that an interpretation $\mathcal{I}$ satisfies an axiom E ($\mathcal{I} \models E$) when:

$$\begin{array}{rcl} C^{\mathcal{I}} \subseteq D^{\mathcal{I}} & if & E = C \sqsubseteq D \ , \\ a^{\mathcal{I}} \in C^{\mathcal{I}} & if & E = a : C \ , \\ \langle a, b \rangle \in R^{\mathcal{I}} & if & E = \langle a, b \rangle : R \ . \end{array}$$

A set of axioms $\mathcal{E}$ is satisfied by an interpretation $\mathcal{I}$ ($\mathcal{I} \models \mathcal{E}$) if $\forall E \in \mathcal{E}$, $\mathcal{I} \models E$. An interpretation $\mathcal{I}$ is a model for an ontology $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ if $\mathcal{I} \models \mathcal{A} \cup \mathcal{T}$. A knowledge base $\mathcal{K}$ is said to entail an axiom E iff $\forall \mathcal{I}$ if $\mathcal{I} \models \mathcal{K}$ then $\mathcal{I} \models E$.

Satisfiability of a $\mathcal{K}$ is a fundamental problem when working with DLs and is often referred to as ontology consistency problem. Besides this problem there are other relevant:

Subsumption Checking is the problem of deciding whether or not $\mathcal{K} \models C \sqsubseteq D$.

Instance Checking is the problem of deciding whether or not $\mathcal{K} \models a : C$.

Concept Consistence is the problem of deciding whether or not for an individual a not occurring in $\mathcal{K}$, $\mathcal{K} \cup \{a : C\}$ has a model.

Instance Retrieval is the problem of finding all instances of a certain concept, *i.e.*, the instance retrieval problem w.r.t. concept C is to determine the set $\{a$ occurs in $\mathcal{K} \mid \mathcal{K} \models a : C\}$.

1.1.3 Concrete Domains

Concrete domains [4, 28, 29] are used to extend DLs in order to deal with concrete datatypes. When dealing with datatypes we assume to have a certain number of ingredients. We assume to have a finite set of data values, a set of elementary datatypes and a set of datatype predicates each of which has a predefined arity $n \geq 1$. A datatype theory is a pair $\mathbf{D} = \langle \Delta^D, \cdot^D \rangle$ where Δ^D is the datatype domain and $\cdot^D$ a mapping that assigns to each data value an element from Δ^D, to each elementary datatype a subset of Δ^D and to each datatype predicate p of arity n a n-ary relation over Δ^D. $\cdot^D$ is extended to all datatype as $\{v_1, ...\}^D = \{v_1^D, ...\}$.

A DL provided with a datatype is denoted adding to its name the label (D) so, $\mathcal{ALC}$ with concrete domains is denoted as $\mathcal{ALC}(D)$. Once we have concrete domains we can define *data properties*. Data properties can be considered as roles that relates an individual to a certain data value. Classical roles are then denoted as object properties. An interpretation $\mathcal{I}$ will map a data property into a subset of $\Delta^{\mathcal{I}} \times \Delta^D$ and will not have an inverse.

1.1.4 $\mathcal{ALC}$

The $\mathcal{AL}$ family is a very basic description logic family. An important increase in terms of expressiveness is $\mathcal{ALC}$. As explained earlier, each letter added to the family name adds a constructor. The $\mathcal{C}$ adds the concept negation constructor *i.e.*, in $\mathcal{ALC}$ negation can be applied too every possible concept, not only to atomic ones, but also to complex ones.

In the following, we say that two concepts C and D are considered equivalent if for each interpretation $\mathcal{I}$ we have that $C^{\mathcal{I}} = D^{\mathcal{I}}$.

We extend the interpretation function of $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ to disjunction, concept negation and qualified existential restriction as follows:

$$\begin{aligned} (C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \ , \\ (\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \ , \\ (\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}. \langle x, y \rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \ . \end{aligned}$$

Then, it can be easily shown that:

$$\begin{aligned} \neg((\neg C) \sqcap (\neg D)) &\equiv C \sqcup D \\ \neg(\forall R.\neg C) &\equiv \exists R.C \ , \end{aligned}$$

where $C \equiv D$ means that C is equivalent to D. Therefore, we can consider $\mathcal{ALC}$ as defined by the following grammar:

$$\begin{array}{rcc} C, D & \rightarrow & A \quad | \\ & & \top \quad | \\ & & \bot \quad | \\ & & \neg C \quad | \\ & & C \sqcap D \quad | \\ & & C \sqcup D \quad | \\ & & \forall R.C \quad | \\ & & \exists R.C \quad . \end{array}$$

Notice that, this time, existential restriction is qualified *i.e.*, a generic concept is allowed as role filler, and a generic concept can be negated. As seen above, the only increase in complexity/expressiveness is brought by the generalization of negation and not by qualification of existential restrictions nor by disjunction.

1.1.5 $\mathcal{SROIQ}$

Another interesting DL languge is $\mathcal{SROIQ}$. This DL has the following features:

$\mathcal{S}$: equivalent to say $\mathcal{ALCR}_+$

$\mathcal{R}_+$: transitive role axioms, denoted as $\mathsf{Trans}(R)$ with semantics:

$$\mathcal{I} \models \mathsf{Trans}(R) \text{ iff } R^{\mathcal{I}} \text{ is transitive} \ .$$

$\mathcal{R}$: complex role inclusion axioms, denoted as $R \circ S \sqsubseteq T$ with semantics:

$$\mathcal{I} \models R \circ S \sqsubseteq T \text{ iff } R^{\mathcal{I}} \circ S^{\mathcal{I}} \subseteq T^{\mathcal{I}} \ .$$

Note that $\mathcal{R}$ is stronger than $\mathcal{R}_+$.

$\mathcal{O}$: Nominals, singleton class, denoted by $\{a\}$, with semantics:

$$\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\} \ .$$

$\mathcal{I}$**:** Inverse roles, denoted with R^-, with semantics

$$(R^-)^{\mathcal{I}} = \{\langle y, x\rangle \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \ .$$

$\mathcal{Q}$**:** Qualified number restriction, denoted by $(\geq n\ R.C)$ and $(\leq n\ R.C)$, with semantics:

$$(\leq n\ R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(R^{\mathcal{I}}(x) \cap C^{\mathcal{I}}(x)) \leq n\ \}$$

$$(\geq n\ R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(R^{\mathcal{I}}(x) \cap C^{\mathcal{I}}(x)) \geq n\ \} \ .$$

$\mathcal{SROIQ}$ ontologies are considered to be of the form $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{R}\rangle$. $\mathcal{T}$ and $\mathcal{A}$ are still a $\mathcal{TBOX}$ and an $\mathcal{ABOX}$, while $\mathcal{R}$ is called an $\mathcal{RBOX}$ and contains all *Roles Inclusion Axioms* (RIAs).

Let us point out that $\mathcal{SROIQ}$ is important as it is the logical counterpart of *OWL 2*, as any *OWL 2* axiom can be mapped into $\mathcal{SROIQ}$. We refer the reader to [19] for further details on $\mathcal{SROIQ}$.

1.1.6 $\mathcal{EL}$ family

The $\mathcal{EL}$ family is an important DL family and will be the family that will be used in the following as the learning language. Besides being the $\mathcal{EL}$ family related to the *OWL 2* profile language OWL EL[34], this family is important because all major reasoning tasks can be performed in polynomial time, while most all other DLs have exponential or higher computational complexity.

An $\mathcal{EL}$ concept is described by the following grammar:

C, D	$\rightarrow$	A	(atomic concept)
		$\top$	(universal concept)
		$C \sqcap D$	(concept conjunction)
		$\exists R.C$	(existential qualified restriction)

$\mathcal{EL}$ has been extended in [2] and successively in [3] to obtain $\mathcal{EL}^{++}(D)$. $\mathcal{EL}^{++}(D)$, as addressed by (D) label, gives the possibility to deal with concrete domains. Besides concrete domains, there are more constructors for concept expressions. The $\mathcal{EL}^{++}(D)$ syntax, as presented in [3], is described in Table 1.1. For the new concept expressions, we extend interpretation function as follows:

$$p(f_1, ..., f_n)^{\mathcal{I}} \ = \ \{x \in \Delta^{\mathcal{I}} \mid \exists y_i \in \Delta^D, f_i(x) = y_i, \langle y_1, ..., y_n\rangle \in p^D, i = 1, ..., n\} \ .$$

C, D	$\rightarrow$	$\top$
		$\bot$
		A
		$\{a\}$
		$C \sqcap D$
		$\exists R.C$
		$p(f_1, ..., f_n)$
E	$\rightarrow$	$C \sqsubseteq D$
		$R_1, ..., R_n \sqsubseteq R$
		$dom(R) \sqsubseteq C$
		$ran(R) \sqsubseteq C$
		$ref(R)$ (reflexivity)
		$a : C$
		$(a, b) : R$

Table 1.1: $\mathcal{EL}^{++}(D)$ syntax.

Moreover we have to extend our notion of satisfaction for axioms. We say that $\mathcal{I} \models E$ if:

$$
\begin{aligned}
\forall \langle x, y \rangle \in R^{\mathcal{I}}.x \in C^{\mathcal{I}} \quad & if \quad E = dom(R) \sqsubseteq C \ , \\
\forall \langle x, y \rangle \in R^{\mathcal{I}}.y \in C^{\mathcal{I}} \quad & if \quad E = ran(R) \sqsubseteq C \ , \\
\forall x \in \Delta^{\mathcal{I}}.\langle x, x \rangle \in R^{\mathcal{I}} \quad & if \quad E = ref(R) \ , \\
\forall x_1, x_{n+1} \in \Delta^{\mathcal{I}}.(\exists x_2, ..., x_n \in \Delta^{\mathcal{I}}. & \\
(R_1(x_1, x_2) \wedge ... \wedge R_n(x_n, x_{n+1}))) & \\
\rightarrow R(x_1, x_{n+1}) \quad & if \quad E = R_1, ..., R_n \sqsubseteq R \ .
\end{aligned}
$$

1.2 OWL 2 Syntax

As explained above *OWL 2* is strongly related to DLs. However it is still a language on its own and has its syntax. Moreover it has particular constructs that increase the usability of the language. *OWL 2* has some fundamental constructs [32]:

IRI: *Internationalized Resource Identifier*, identifiers associated to resources;

Declarations: used to declare what a certain entity is;

Assertions: used to assert something. They can be compared to DL axioms;

Annotations: used to add verbose information to IRIs and assertions.

IRIs are used to identify resources. Resources are intended to be all basic structures named in ontologies. These basic structures are called **entities** and are divided into:

- Classes: represent atomic concept;
- Individuals: ontology individuals;
- ObjectProperties: represent roles that involve two individuals;
- DataProperties: represent roles that associate a concrete data value to an individual;
- Datatypes: represent a particular concrete datatype pre-defined, *primitive*, or defined through some constructors, *complex*;
- Other structures, *e.g.*, AnnotationProperty.

Each entity must be declared through a declaration. Both IRI and declarations can be annotated.

Declarations, assertions and annotations can be added to an ontology through axioms. An *OWL 2* ontology is composed of axioms. Below some examples of *OWL 2* assertions are given.

- `AnnotationAssertion(<AnnotationProperty> <IRI/Axiom> <AnnotationValue>)`
  Used to associate an IRI/Axiom to an AnnotationProperty with value AnnotationValue. Annotation properties are labels that identifies particular properties. Some properties are defined in rdfs, *e.g.*, `rdfs:label`.
- `ClassAssertion(<ClassExpression> <Individual>)`
  Used to declare that Individual belongs to ClassExpression.
- `ObjectPropertyAssertion(<ObjectProperty> <Individual1> <Individual2>)`
  Used to say that Individual1 and Individual2 are related through ObjectProperty.

- `DataPropertyAssertion(<DataProperty> <Individual> <DataValue>)`
  Used to say that Individual and DataValue are related through DataProperty, *i.e.*, Individual has DataProperty DataValue.

- `SubClassOf(<ClassExpression1> <ClassExpression2>`
  Used to say that ClassExpression1 is a subclass of ClassExpression2.

Many more axioms can be defined, a full list can be found in [32].
Finally, we recall some class expressions we may have in *OWL 2* [32].

- `ObjectIntersectionOf(<ClassExpression1> <ClassExpression2>)`
  Defines the conjunction of two class expressions;

- `ObjectUnionOf(<ClassExpression1> <ClassExpression2>)`
  Defines the disjunction of two class expressions;

- `ObectComplementOf(<ClassExpression>)`
  Defines the complement of a class expression;

- `ObjectSomeValuesFrom(<ObjectProperty> <ClassExpression>)`
  Defines an existential restriction on ObjectProperty qualified with ClassExpression;

- `DataSomeValuesFrom(<DataProperty> <Datatype>)`
  Defines an existential restriction on DataProperty qualified with Datatype.

Similar constructions can be applied to object properties, data properties and datatypes. The complete *OWL 2* syntax can be found in [32].

Chapter 2

Fuzzy Logics and Fuzzy *OWL*

When dealing with fuzziness we should first point out the difference between uncertainty and vagueness. In knowledge representation and artificial intelligence there has been a long-lasting misunderstanding between the two. An accurate explanation of this distinction can be found in [8]. A simple example to rapidly explain the difference between uncertainty and vagueness is the following. Assume we want to say whether or not the assertion ‘‘today is a hot day" is true. This assertion involves the fuzzy concept hot. Using crisp logic (boolean valued) we should set a threshold and say, for instance, that if the temperature is lower than 24° C then it is not a hot day and, it is hot, if temperature is equal or higher than 24° C. Then if we have 23.9° C we do not have a hot day. If, instead, we have 24° C, the day would be considered hot. This does not fit with the common intuition of the concept of "hot day". To deal with this issue we can use fuzzy sets/concepts[38]. We can define a fuzzy function that associates to a given temperature a certain degree of truth. So, we say that hot day is a fuzzy concept.

If instead we say that *"tomorrow we will have at least* 24° *C"* then we involve uncertainty because we may not know exactly which temperature we will have tomorrow.

Summing up, when dealing with vagueness we use fuzzy theory to say *how much* an individual belongs to a certain set/concept. When dealing with uncertainty we use probability/possibility theory to say *how probable/possible* a certain individual belongs to a set/concept. There's also a third option to consider when we deal with both uncertainty and vagueness. Considering previous example, if we say *"tomorrow it will be hot"* we should have a probability/possibility distribution defined over fuzzy values. This way we involve both vagueness, with the concept ‘hot’ day, and uncertainty, as we try to predict tomorrow's temperature. In this case we need both fuzziness and probability/possibility.

In this work we will only deal with fuzzy logic.

Before starting to describe what is fuzziness and how it is defined, it is necessary to explain why it is useful. Fuzziness increases not only understandability but it also makes the description given by knowledge bases closer to common intuition. For instance, it is questionable saying that a person who is 179 cm tall is not tall and that a person who is 180 cm tall is tall. Fuzzy set theory have been developed to overcome such limitations.

2.1 Fuzzy Sets Basics

Before starting to talk about fuzzy logics let's have a small introduction to fuzzy sets theory. Fuzzy theory is based on a simple modification of classical set theory. Classical set theory has as first order citizen the *characteristic function*. Assuming we have a universe $\mathcal{X}$, we define a function μ_A as the characteristic function of set $A \subseteq \mathcal{X}$ as $\mu_A : \mathcal{X} \to \{0, 1\}$. The intuition of characteristic function is that $\forall x \in \mathcal{X}, \mu_A(x) = 1$ iff $x \in A$.

When dealing with fuzziness we change the definition of characteristic function to $\mu_A : \mathcal{X} \to [0, 1]$ with the intuition that $\forall x \in \mathcal{X}$, x belongs to A with degree $\mu_A(x)$.

Another thing to focus on is how we can define membership. Let's consider the fuzzy concept tall. We can define a membership function that says whoever is at least 190 cm is tall with degree 1, whoever is 170 cm tall or less is tall with degree 0 and building the function by linearly interpolating the values between 170 and 180. Then we can define a linear function that is 0 in 170 and 1 in 190. This way we have a definition of fuzzy concept tall. So if someone is 180 cm tall he will be tall with degree 0.5.

$$\mu_{tall}(x) = \begin{cases} 1 & \text{if } height(x) \geq 190 \\ 0 & \text{if } height(x) \leq 160 \\ \frac{x-170}{190-170} & otherwise \ , \end{cases}$$

where $height(x)$ is the height of x. This kind of function is called right shoulder and it is a typical fuzzy function. Typical fuzzy function are listed on Table 2.1.

2.1.1 Norm-Based Fuzzy Set Operations

Among set operations that we can use on fuzzy sets *norm-based operations* are among the most widely used. These kind of operations are called *t-norms*, *t-conorms* or *s-norms* and *negation*. T-norms, denoted as $\otimes$, replace conjunction, s-norms, denoted as $\oplus$, replace disjunction and negation, denoted

Name	Denotation	Definition	Picture
Right Shoulder	$rs(a,b)$	$rs(a,b)(x) = \begin{cases} 1 & \text{if } x \geq b \\ 0 & \text{if } x \leq a \\ \frac{x-a}{b-a} & otherwise \end{cases}$	Figure 2.1(d)
Left Shoulder	$ls(a,b)$	$ls(a,b)(x) = \begin{cases} 1 & \text{if } x \leq a \\ 0 & \text{if } x \geq b \\ \frac{b-x}{b-a} & otherwise \end{cases}$	Figure 2.1(c)
Triangular(b)	$tri(a,b,c)$	$tri(a,b,c)(x) = \begin{cases} 0 & \text{if } x \leq a \\ 0 & \text{if } x \geq c \\ 1 & \text{if } x = b \\ \frac{x-a}{b-a} & \text{if } a < x < b \\ \frac{c-x}{c-b} & \text{if } b < x < c \end{cases}$	Figure 2.1(b)
Trapezoidal	$trz(a,b,c,d)$	$trz(a,b,c,d)(x) = \begin{cases} 0 & \text{if } x \leq a \\ 0 & \text{if } x \geq d \\ 1 & \text{if } b \leq x \leq c \\ \frac{x-a}{b-a} & \text{if } a < xb \\ \frac{d-x}{d-c} & \text{if } c < x < d \end{cases}$	Figure 2.1(a)

Table 2.1: Typical fuzzy functions.

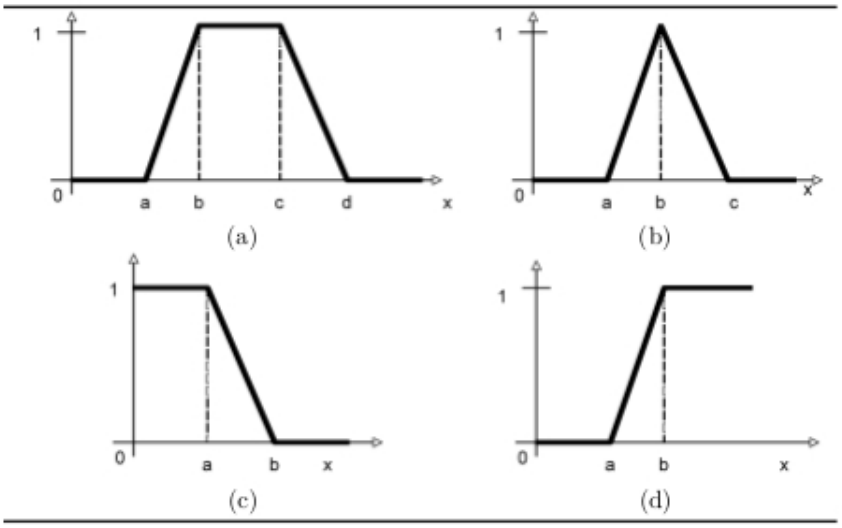

Figure 2.1: (a) Trapezoidal Function: *trz(a,b,c,d)* (b) Triangular Function: *tri(a,b,c)* (c) Left Shoulder: *ls(a,b)* (d) Right Shoulder: *rs(a,b)*.

as $\ominus$, replace, boolean negation. Considering three operations $\langle \otimes, \oplus, \ominus \rangle$ they can be considered, respectively t-norm, s-norm and negation if they respects property of Table 2.2

Axiom Name	T-norm	S-norm
Boundary Condition	$a \otimes 1 = a$	$a \oplus 0 = a$
Commutativity	$a \otimes b = b \otimes a$	$a \oplus b = b \oplus a$
Associativity	$(a \otimes b) \otimes c = a \otimes (b \otimes c)$	$(a \oplus b) \oplus c = a \oplus (b \oplus c)$
Monotonicity	if $b \leq c$ then $a \otimes b \leq a \otimes c$	if $b \leq c$ then $a \oplus b \leq a \oplus c$

Axiom Name	Negation
Boundary Condition	$\ominus 0 = 1$
	$\ominus 1 = 0$
Antitonicity	if $a \leq b$ then $\ominus a \geq \ominus b$

Table 2.2: Axioms for norm-based operations.

A useful requirement is *duality* between t-norm and s-norm.

Definition 1. *Two operations $\otimes$ and $\oplus$ are said to be dual if*

$$a \otimes b = 1 - ((1 - a) \oplus (1 - b))$$

Duality guarantees that properties of Table 2.2 are respected as stated in Proposition 1.

Proposition 1 ([21])**.** *Let $\otimes$ be a t-norm then an operation $\oplus$ dual to $\otimes$ is an s-norm*

Another fuzzy operation usually adopted is *implication*. Fuzzy implication replaces boolean implication. An implication operator is often represented as $\Rightarrow$. It has to respect axioms of Table 2.3.

Usually implication is asked to be an *r-implication* w.r.t. a certain t-norm.

Definition 2. *An operation $\Rightarrow$ is said to be an* r-implication *w.r.t. a t-norm $\otimes$ if*

$$a \Rightarrow b = sup\{c \mid a \otimes c \leq b\}$$

Axiom Name	Axiom
Boundary Condition	$0 \Rightarrow b = 1$
	$a \Rightarrow 1 = 1$
Antitonicity	if $a \leq b$ then $a \Rightarrow c \geq b \Rightarrow c$
Monotonicity	if $b \leq c$ then $a \Rightarrow b \leq a \Rightarrow c$

Table 2.3: Fuzzy Implication Axioms.

Name	T-Norm $x \otimes y$	S-Norm $x \oplus y$	Implication $x \Rightarrow y$	Negation $\ominus x$
Łukasiewicz	$max(x+y-1,0)$	$min(x+y,1)$	if $x \leq y$ then 1 else $1-x+y$	$1-x$
Gödel	$\min(x,y)$	$max(x,y)$	if $x \leq y$ then 1 else y	if $x == 0$ then 1 else 0
Product	$x \cdot y$	$(x+y)-x \cdot y$	if $x \leq y$ then 1 else y/x	if $x == 0$ then 1 else 0
Zadeh	$\min(x,y)$	$max(x,y)$	$max(1-x,y)$	$1-x$

Table 2.4: Fuzzy Operations Families.

An r-implication respect axiom of Table 2.3.
Some families of fuzzy operations are described in Table 2.4

2.1.2 Fuzzy Cardinality

When working with fuzzy sets we have to reconsider also the notion of cardinality. Informally, with classic sets the cardinality was defined as the number of elements belonging to a set. This definition is no more valid on fuzzy sets as we do not have a boolean definition of membership. Many works have been developed on this topic, so many different ways of defining fuzzy cardinality have been proposed.

There are three fundamental groups of cardinalities of fuzzy sets [10]:

- *Scalar Cardinalities*: cardinalities that uses a single number (scalar) to express cardinality;
- *Fuzzy Cardinalities*: cardinalities that uses a fuzzy set to denote cardinality.
- *Gradual Numbers*: cardinalities that uses gradual numbers defined by Dubois and Prade ([6]).

For our purpose we use a single real value indicating cardinality so we prefer using scalar cardinalities. However we could also have adopted fuzzy cardinalities beside with a defuzzification method, *e.g.*, middle of maxima.

A classic cardinality is *Sigma Count*. This cardinality sums up the degree of truth of all elements belonging to the universe. This sum is the cardinality of the set.
Let $\mathcal{X}$ be the universe and $B \subseteq \mathcal{X}$ a set. The cardinality of B w.r.t. $\mathcal{X}$ is:

$$card_{\mathcal{X}}(B) = \sum_{x \in \mathcal{X}} \mu_B(x) \ .$$

Note that even if we have a lot of elements belonging to B with very low degree we can have a high value for the cardinality.

A complete list of cardinalities is available in [10].

2.2 Fuzzy Logics Basics

After having introduced the idea of fuzziness we can apply it to logics. In fuzzy logic statement is not necessarily true or false but every statement has a degree of truth that states how much a statement is true.

A fuzzy statement can be considered as a pair $\langle \phi, r \rangle$ where ϕ is a FOL statement and r a real value in $[0, 1]$. The statement is interpreted as ϕ *is true with degree at least* r. ϕ can be described by the grammar:

$$\begin{array}{rcrcl} \phi, \psi & \rightarrow & A(x) & | & (Atomic) \\ & & \phi \wedge \psi & | & (Conjunction) \\ & & \phi \vee \psi & | & (Disjunction) \\ & & \phi \rightarrow \psi & | & (Implication) \\ & & \phi \leftrightarrow \psi & | & (Equivalence) \\ & & \neg\phi & | & (Negation) \\ & & \exists x.\phi & | & (ExistentialRestriction) \\ & & \forall x.\phi & & (UniversalRestriction) \end{array}$$

To define interpretations we need a family of fuzzy operations. Then for an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ we have:

$$\begin{array}{rcl} \mathcal{I}(A(x)) & = & A^{\mathcal{I}}(\mathcal{I}(x)) \ , \\ \mathcal{I}(\phi \wedge \psi) & = & \mathcal{I}(\phi) \otimes \mathcal{I}(\psi) \ , \\ \mathcal{I}(\phi \vee \psi) & = & \mathcal{I}(\phi) \oplus \mathcal{I}(\psi) \ , \\ \mathcal{I}(\phi \rightarrow \psi) & = & \mathcal{I}(\phi) \Rightarrow \mathcal{I}(\psi) \ , \\ \mathcal{I}(\phi \leftrightarrow \psi) & = & (\mathcal{I}(\phi) \Rightarrow \mathcal{I}(\psi)) \otimes (\mathcal{I}(\psi) \Rightarrow \mathcal{I}(\phi)) \ , \\ \mathcal{I}(\neg\phi) & = & \ominus \mathcal{I}(\phi) \ , \\ \mathcal{I}(\exists x.\phi) & = & \sup\limits_{a \in \Delta^{\mathcal{I}}} \mathcal{I}_x^a(\phi) \ , \\ \mathcal{I}(\forall x.\phi) & = & \inf\limits_{a \in \Delta^{\mathcal{I}}} \mathcal{I}_x^a(\phi) \ , \end{array}$$

where $\mathcal{I}(x) \in \Delta^{\mathcal{I}}$, $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow [0, 1]$ and $\mathcal{I}_x^a$ is the the same as $\mathcal{I}$ but $I(x) = a$.

We say that $\mathcal{I}$ is a model of (satisfies) a fuzzy statement $\langle \phi, r \rangle$, $\mathcal{I} \models \langle \phi, r \rangle$ iff $\mathcal{I}(\phi) \geq r$. We say that two formulas ϕ, ψ are equivalent, denoted $\phi \equiv \psi$, iff for each interpretation $\mathcal{I}$, $\mathcal{I}(\phi) = \mathcal{I}(\psi)$. We say that an interpretation $\mathcal{I}$ satisfies a formula ϕ, denoted $\mathcal{I} \models \phi$ iff $\mathcal{I}(\phi) = 1$. We say that a formula is a tautology iff it is satisfied by every interpretation. Defined a *fuzzy knowledge*

base $\mathcal{K}$ as a set of fuzzy statements, we say that an interpretation $\mathcal{I}$ satisfies $\mathcal{K}$, denoted $\mathcal{I} \models \mathcal{K}$, iff $\mathcal{I}$ satisfies each axiom in $\mathcal{K}$. We say that a fuzzy statement $\langle \phi, r \rangle$ is a logical consequence of a fuzzy knowledge base $\mathcal{K}$ iff $\forall \mathcal{I}, \mathcal{I} \models \mathcal{K}$ implies $\mathcal{I} \models \langle \phi, r \rangle$.

Now we can define two problems related to fuzzy knowledge bases:

1. **Best Entailment Degree:** $bed(\mathcal{K}, \phi) = sup\{r \mid \mathcal{K} \models \langle \phi, r \rangle\}$
2. **Best Satisfiability Degree:** $bsd(\mathcal{K}, \phi) = \sup_{\mathcal{I}} \{\mathcal{I}(\phi) \mid \mathcal{I} \models \mathcal{K}\}$

2.2.1 Witnessed Models

We say that a fuzzy interpretation $\mathcal{I}$ is a *witnessed interpretation* iff

$$\mathcal{I}(\exists x.\phi) = \mathcal{I}_x^a(\phi), \text{ for some } a \in \Delta^{\mathcal{I}}$$

$$\mathcal{I}(\forall x.\phi) = \mathcal{I}_x^a(\phi), \text{ for some } a \in \Delta^{\mathcal{I}}$$

As can be noticed from the definition of fuzzy interpretation, we can have interpretations without a witness. Considering that $\Delta^{\mathcal{I}} = \mathcal{N}$ and that $A^{\mathcal{I}}(n) = 1 - \frac{1}{n}$. We have that:

$$\mathcal{I}(\exists x.A(x)) = \sup_n \mathcal{I}_x^n(A(x)) = \sup_n 1 - \frac{1}{n} = 1$$

so $\mathcal{I}(\exists x.A(x)) = 1$ while there is no $a \in \Delta^{\mathcal{I}}$ for which $\mathcal{I}_x^a(A(x)) = 1$.

It can be shown that [14, 15, 16, 17]:

Proposition 2. *Under Łukasiewicz logic, a fuzzy statement has a fuzzy model iff it has a witnessed fuzzy model.*

Proposition 2 does not hold for Gödel logic and Product logic.

2.2.2 Fuzzy Description Logics

Generally, when we want to use Fuzzy DLs we define Fuzzy DL statements of the form $\langle E, n \rangle$ where E is a DL axiom and n is the degree stating how much E is true. The axiom is then mapped into a FOL statement ϕ. So $\langle E, n \rangle$ can be treated as a Fuzzy FOL statement $\langle \phi, n \rangle$.

However it is preferable to avoid such translation and trying to find a direct way to work with Fuzzy Description Logics.

2.2.2.1 Syntax and Semantics

Working with $\mathcal{SROIQ}$ we have that possible Fuzzy DL statements are:

1. $\langle a : C, n\rangle$: *Fuzzy Concept Assertions*, a is an individual and C a concept expression, it states that a is C with degree at least n;

2. $\langle (a, b) : R, n\rangle$: *Fuzzy Role Assertions*, a and b are individual and R is a role, it states that a and b are related through R with degree at least n;

3. $\langle C \sqsubseteq D, n\rangle$: *Fuzzy GCIs*, C and D are concept expressions, it states that B is a subclass of C with degree at least n;

4. $\langle R_1 ... R_m \sqsubseteq R, n\rangle$: *Fuzzy RIAs*, where $R_1, ..., R_m, R$ are roles, it states that the composition of $R_1, ..., R_m$ is a subrole of R with degree at least n;

5. $(w_1 \cdot A_1 + ... + w_n \cdot A_n) \sqsubseteq C$: *Weighted Sum*, where $w_1, ..., w_n$ are real numbers in $[0, 1]$ such that $\sum_{i=1}^{n} w_i <= 1$, $A_1, ..., A_n, WeightedSum$ are atomic concepts;

6. Axioms $\mathsf{trans}(R), \mathsf{disj}(S_1, S_2), \mathsf{ref}(R), \mathsf{irr}(S), \mathsf{sym}(R)$ and $\mathsf{asy}(S)$, these axioms are not fuzzy and so the interpretations satisfying such axioms must behave as for crisp case.

An interpretation is considered as $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\rangle$ where $\Delta^{\mathcal{I}}$ is an *interpretation domain* and $\cdot^{\mathcal{I}}$ an *interpretation function* that maps an atomic concept into a function from $\Delta^{\mathcal{I}}$ to $[0, 1]$, *i.e.*, $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \to [0, 1]$. Each role is mapped into a function $R^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \to [0, 1]$. Finally an individual is mapped into an element of interpretation domain, *i.e.*, $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

To extend interpretation definition we need a family of fuzzy operators. Assume that $\langle \otimes, \oplus, \Rightarrow, \ominus\rangle$ are a family of fuzzy operators then:

$$\begin{aligned}
\mathcal{I}(A)(x) &= A^{\mathcal{I}}(x) \ , \\
\mathcal{I}(C \sqcap D)(x) &= \mathcal{I}(C)(x) \otimes \mathcal{I}(D)(x) \ , \\
\mathcal{I}(C \sqcup D)(x) &= \mathcal{I}(C)(x) \oplus \mathcal{I}(D)(x) \ , \\
\mathcal{I}(\neg C)(x) &= \ominus \mathcal{I}(C)(x) \ , \\
\mathcal{I}(\forall R.C)(x) &= \inf_{y \in \Delta^{\mathcal{I}}} \{R^{\mathcal{I}}(x, y) \Rightarrow \mathcal{I}(C)(y)\} \ , \\
\mathcal{I}(\exists R.C)(x) &= \sup_{y \in \Delta^{\mathcal{I}}} \{R^{\mathcal{I}}(x, y) \otimes \mathcal{I}(C)(y)\} \ .
\end{aligned}$$

Then we define the notion of satisfaction for fuzzy statements *i.e.*, $\mathcal{I} \models E$ when:

$$\begin{aligned}
\mathcal{I}(C)(a^{\mathcal{I}}) \geq n \quad & if \quad E = \langle a : C, n \rangle \ , \\
R^{\mathcal{I}}(a^{\mathcal{I}}) \geq n \quad & if \quad E = \langle (a,b) : R, n \rangle \ , \\
(inf_{x \in \Delta^{\mathcal{I}}} \mathcal{I}(C)(x) \Rightarrow \mathcal{I}(D)(x)) \geq n \quad & if \quad E = \langle C \sqsubseteq D, n \rangle \ , \\
(inf_{x_1, x_{m+1} \in \Delta^{\mathcal{I}}} \ (sup_{x_2, \ldots, x_m \in \Delta^{\mathcal{I}}} & \\
(R_1^{\mathcal{I}}(x_1, x_2) \otimes \ldots \otimes R_m^{\mathcal{I}}(x_m, x_{m+1})) & \\
\Rightarrow R^{\mathcal{I}}(x_1, x_{m+1}))) \geq n \quad & if \quad E = \langle R_1 \ldots R_m \sqsubseteq R, n \rangle \ , \\
(inf_{x \in \Delta^{\mathcal{I}}} \ (\textstyle\sum_{i=1}^{n} w_i \cdot \mathcal{I}(A_i)(x)) \Rightarrow \mathcal{I}(C)(x)) = 1 & \\
if \ E = (w_1 \cdot A_1 + \ldots + w_n \cdot A_n) \sqsubseteq C \quad & .
\end{aligned}$$

Further definitions are needed to cover other type of axioms. However as they will not be used in the following they are omitted. The interested reader can find a complete definition in [37].

A knowledge base is defined as $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{T}$ is the fuzzy $\mathcal{TBOX}$, $\mathcal{A}$ is the fuzzy $\mathcal{ABOX}$ and $\mathcal{R}$ is the fuzzy $\mathcal{RBOX}$.

We say that an interpretation $\mathcal{I}$ satisfies a fuzzy DL statement $\langle E, n \rangle$ if $\mathcal{I}(E) >= n$. An interpretation $\mathcal{I}$ is said to satisfy a set of fuzzy DL statements if it satisfies each statement of the set. An interpretation $\mathcal{I}$ is said to satisfy a fuzzy DL ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{R} \rangle$ iff $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \mathcal{A}$ and $\mathcal{I} \models \mathcal{R}$. A fuzzy statement $\langle E, r \rangle$ is said to be a logical consequence of an ontology $\mathcal{K}$ ($\mathcal{K} \models \langle E, r \rangle$) iff each model of $\mathcal{K}$ is a model for $\langle E, r \rangle$. As before we can define:

Best Entailment Degree: $bed(\mathcal{K}, E) = sup\{r \mid \mathcal{K} \models \langle E, r \rangle\}$;

Best Satisfiability Degree: $bsd(\mathcal{K}, E) = \sup_{I}\{E^{\mathcal{I}} \mid \mathcal{I} \models \mathcal{K}\}$;

2.2.2.2 Concrete Domains

Like in classical DLs, in fuzzy DLs we may have *concrete domains*. However, also if it is possible, it is a little bit harder to consider them, as, when working under fuzziness, we already put together logic and concrete values to represent degree of truth. It is very important to distinguish between concrete data values and concrete values used to denote degrees. Anyway concrete domains work exactly as for classical DLs.

For our description of fuzzy concrete domains we refer to [36]. A *fuzzy concrete domain*, also called *fuzzy datatype theory*, is a pair $\mathbf{D} = \langle \Delta^D, \cdot^D \rangle$ where Δ^D is called datatype domain and $\cdot^D$ is a mapping that assigns to each data value an element of Δ^D and to each n-ary datatype predicate an n-ary fuzzy relation over Δ^D. As fuzzy DLs support unary datatypes only, then $\cdot^D$ maps each datatype predicate into a function from Δ^D to $[0, 1]$.

Typically datatype predicates $\mathbf{d}$ are the already mentioned fuzzy functions

$$\mathbf{d} \rightarrow ls(a,b) \mid rs(a,b) \mid tri(a,b,c) \mid trz(a,b,c,d) \ ,$$

and crisp functions

$$\mathbf{d} \rightarrow \geq_v \ \mid \ \leq_v \ \mid \ =_v \ .$$

Introducing concrete domains we can have data properties besides object properties. An interpretation $\mathcal{I}$ will map an *object property* into a function from $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \rightarrow [0,1]$ and a *data property* into a function from $\Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}} \rightarrow [0,1]$. Moreover *unique name assumption* (UNA) will be adopted, *i.e.*, $v_1^{\mathcal{I}} \neq v_2^{\mathcal{I}}$ if $v_1 \neq v_2$. From now on, datatatypes will be denoted as $\mathbf{d}$ and concrete individuals will be denoted as v or v_i.

Finally, concrete domains can be used in concept expressions as follows:

$$C \rightarrow \forall T.\mathbf{d} \mid \exists T.\mathbf{d} \ ,$$

where T is a data property and $\mathbf{d}$ a datatype. For what concerns us only $\exists T.\mathbf{d}$ will be used.

2.3 Fuzzy *OWL 2*

Fuzzy *OWL 2* is a standard presented in [12]. The standard has been defined on *OWL 2* in such a way *OWL 2* reasoners still work on fuzzy *OWL 2* ontologies and fuzzy reasoners can find fuzzy informations. The most natural way to do so is to define fuzziness through *OWL* annotations. So each fuzzy definition is given as an annotation. This solution permits the *OWL 2* standard reasoner to ignore fuzzy structures. However fuzzy *OWL 2* reasoners can find fuzzy informations and work with them.

All fuzzy definitions are given through an annotation property called `fuzzyOWL2`. This annotation property has a field called `fuzzyType` which indicates which type of fuzzy annotation we are defining. `fuzzyType` cha be:

Concept: when we are defining some type of fuzzy concept;

Datatype: when we are defining a fuzzy datatype;

Role: when we want to define a modifier, *e.g.*, `very`;

Axiom: when we are defining a fuzzy axiom, *i.e.*, an axiom having a degree of truth;

Ontology: added to the ontology to state that we are working on a fuzzy ontology with some family of fuzzy operators.

Fuzzy annotations has, as annotation value, plain text that has xml-like format. This format can be understood by fuzzy *OWL* reasoners. Here we have some examples of fuzzy structures:

Fuzzy Datatype: Suppose we have a datatype with IRI datatypeIRI, then we can annotate it with:

```
< fuzzyOwl2 fuzzyType ="datatype" >
  < DATATYPE >
< /fuzzyOwl2 >
```

where:

```
< DATATYPE >:=
  < Datatype type="leftshoulder" a="<DOUBLE>" b ="<DOUBLE>" /> |
  < Datatype type="rightshoulder" a="<DOUBLE>" b="<DOUBLE>" /> |
  < Datatype type="triangular" a="<DOUBLE>" b="<DOUBLE>"
    c="<DOUBLE>" /> |
  < Datatype type="trapezoidal" a="<DOUBLE>" b="<DOUBLE>"
    c="<DOUBLE>" d="<DOUBLE>" />
```

Fuzzy SubClassOf Axiom: Suppose we want to define a fuzzy `SubClassOf` axiom, then we can annotate the axiom with a fuzzy label stating its degree of truth. When the SubClassOf axiom is defined, it should be annotated with the following annotation:

```
<fuzzyOwl2 fuzzyType ="axiom" >
  < Degree value ="<DOUBLE>" />
</fuzzyOwl2 >
```

where `< DOUBLE >` is a double number.

Fuzzy Weighted Sum: Fuzzy weighted sum

$$(0.1 \cdot B1 + 0.2 \cdot B2 + 0.1 \cdot B3 + 0.2 \cdot B4 + 0.4 \cdot B5) \sqsubseteq weightedSum$$

can be expressed with the following annotation:

```
< fuzzyOwl2 fuzzyType ="concept" >
  < Concept type ="weightedSum" >
    < Concept type ="weighted" value ="0.1" base="B1" />
    < Concept type ="weighted" value ="0.2" base="B2" />
    < Concept type ="weighted" value ="0.1" base="B3" />
    < Concept type ="weighted" value ="0.2" base="B4" />
    < Concept type ="weighted" value ="0.4" base="B5" />
  </Concept >
< /fuzzyOwl2 >
```

Part II

Learning Fuzzy $\mathcal{EL}$(D) Inclusion Axioms from Crisp OWL Ontologies

Chapter 3

Axiom Learning Introduction

We will first introduce the problem of axiom learning and what we use to face it. Once the general setting has been introduced we will present our algorithms and finally we will compare them.

3.1 Axiom Learning

Axiom learning is a field of machine learning. It can be considered as the problem of finding rules that define a certain class. Axiom learning setting can be defined as follows:

Definition 3 (Axiom Learning)**.** *Suppose we have a knowledge base $\mathcal{K}$. Let* Ind *be the set of individual appearing in $\mathcal{K}$ and let T be a concept expression definable in $\mathcal{K}$. Axiom learning is the problem of finding a set of rules $H = \{R_1, ..., R_n\}$ such that:*

- $\mathcal{K} \nvDash R_i$ *with* $i = 1, ..., n$*;*
- $\mathcal{K} \cup H \nvDash \bot$*;*
- $\mathcal{K} \models a : T \Rightarrow (\mathcal{K} \backslash T) \cup H \models a : T$ *for each* $a \in \mathsf{Ind}$*.*

where $\mathcal{K}\backslash T$ is $\mathcal{K}$ removing concept T and all axioms that involve such concept.

Notice that implicitly we ask that learnt rules cannot state that an element belonging to T also belong to $\neg T$, *i.e.*, $\mathcal{K} \models a : \neg T \Rightarrow (\mathcal{K} \backslash T) \cup H \nvDash a : T$ for some $a \in \mathsf{Ind}$. In fact if previous statement does not stand then $\mathcal{K} \cup H \models \bot$.

Our setting is based on Crisp (not fuzzy) *OWL* ontologies. So, in our case $\mathcal{K}$ is a crisp *OWL* ontology. Moreover T is an atomic concept, *i.e.*, an *OWL*

class. Each R_i is a SubClassOf axiom of the type $C \sqsubseteq T$. Where C is an $\mathcal{EL}(D)$ class expression and so:

$$\begin{array}{rcll} C & \rightarrow & A & | \\ & & C \sqcap D & | \\ & & \exists R.C & | \\ & & \exists S.\mathbf{d} \ , & \end{array}$$

$$\begin{array}{rcll} \mathbf{d} & \rightarrow & ls(a,b) & | \\ & & rs(a,b) & | \\ & & tri(a,b,c) & | \\ & & trz(a,b,c,d) \ , & \end{array}$$

where D is an $\mathcal{EL}(D)$ class expression, R is an object property, S is a data property and d a fuzzy datatype.

3.1.1 Computing fuzzy datatypes

When there is a data property available it is possible to adopt a discretization method to partition it into a finite number of sets. After such a discretization has been obtained we can define a fuzzy function on each set and obtain a fuzzy partition.

In this work we adopted two simple discretization algorithms. We used *equal width triangular partition* and *equal width trapezoidal partition* over numerical data values.

The first algorithms looks for minimum and maximum value of a data property S into the ontology $\mathcal{K}$. Minimum value of a data property is $min = inf\{v \mid \mathcal{K} \models (a, v) : S \text{ for some } a \in \mathsf{Ind}_{\mathcal{K}}\}$ and maximum is $max = sup\{v \mid \mathcal{K} \models (a, v) : S \text{ for some } a \in \mathsf{Ind}_{\mathcal{K}}\}$. Then interval $[min; max]$ is split into $n > 1$, intervals of the same width. Each of the n intervals, $I_1, ..., I_n$, has a fuzzy function associated. Let $\Delta = \dfrac{max - min}{n - 1}$ then I_1 has associated a left shoulder $ls(min, min + \Delta)$, I_n instead has associated a right shoulder $rs(max - \Delta, max)$ and I_i with $i = 2, ..., n - 1$, has associated a triangular function $tri(min + (i - 2) \cdot \Delta, min + (i - 1) \cdot \Delta, min + i \cdot \Delta)$.

If instead we use the second algorithm the only difference is that I_i with $i = 2, ..., n - 1$, has associated a trapezoidal function $trz(min + (i - 2) \cdot \Delta, min + (i - 1) \cdot \Delta, min + i \cdot \Delta, min + (i + 1) \cdot \Delta)$.

Example 1. *Suppose we want to discretize has_price and we have that maximum value for has_price is 150 and minimum value is 30. Suppose we want to have five intervals. With triangular equal width we will have (Figure 1(a)):*

1. $ls(30, 60)$;
2. $tri(30, 60, 90)$;
3. $tri(60, 90, 120)$;
4. $tri(90, 120, 150)$;
5. $rs(120, 150)$.

Instead with trapezoidal we will have (Figure 1(b)):

1. $ls(30, 60)$;
2. $trz(30, 60, 90, 120)$;
3. $trz(60, 90, 120, 150)$;
4. $trz(90, 120, 150, 180)$;
5. $rs(120, 150)$.

3.1.2 Open World and Closed World Assumptions

Another thing to focus on is the difference between Open World Assumption (OWA) and Closed World Assumption (CWA).

Definition 4 (Open World Assumption)**.** *Let $\mathcal{K}$ be an* OWL 2 *ontology, A be an atomic concept expression definable in $\mathcal{K}$ and a an individual from $\mathcal{K}$. If open world is assumed we have that:*

- $\mathcal{K} \models a : A$ *iff for each interpretation* $\mathcal{I}$ *we have* $\mathcal{I} \models \mathcal{K}$ *then* $\mathcal{I} \models a : A$
- $\mathcal{K} \models a : \neg A$ *iff for each interpretation* $\mathcal{I}$ *we have* $\mathcal{I} \models \mathcal{K}$ *then* $\mathcal{I} \models a : \neg A$

Definition 5 (Closed World Assumption)**.** *Let $\mathcal{K}$ be an* OWL 2 *ontology, A be an atomic concept expression definable in $\mathcal{K}$ and a an individual from $\mathcal{K}$. If open world is assumed we have that:*

- $\mathcal{K} \models_{\{CWA\}} a : A$ *iff for each interpretation* $\mathcal{I}$ *we have* $\mathcal{I} \models \mathcal{K}$ *then* $\mathcal{I} \models a : A$
- $\mathcal{K} \models_{\{CWA\}} a : \neg A$ *iff there exists an interpretation* $\mathcal{I}$ *s.t.* $\mathcal{I} \models \mathcal{K}$ *then* $\mathcal{I} \not\models a : A$

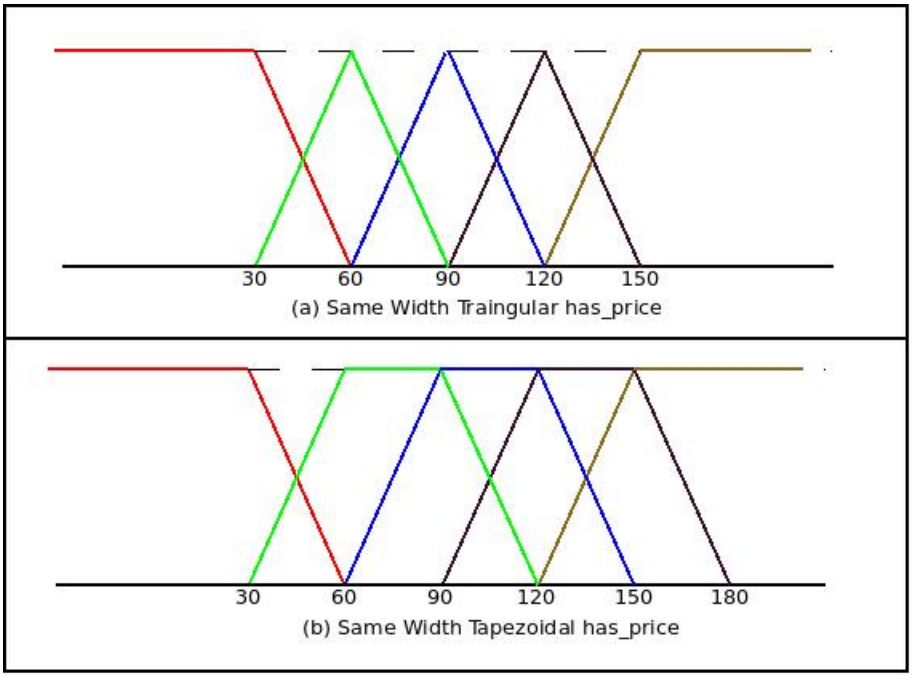

Figure 3.1: Trapezoidal and Triangular partitions of hasPrice from Example 1.

Intuitively if we assume OWA, an individual belongs to an atomic concept's complement if and only if we can prove that. Assuming CWA if an individual cannot be proven to belong to a certain atomic concept then it is assumed to belong to its complement.

The difference between the two assumptions relies in negative classification. In our case this distinction is very useful when building samples sets. In fact if we assume OWA we will have that positives samples are all the individuals that can be proven belonging to target class, negative samples instead will be all the individuals that can be proven belonging to target complement.

If using CWA positive samples will be still those samples that can be proven belonging to target class. Negative samples will be all the others.

3.2 General Settings

Summing up in our setting we have a crisp *OWL 2* ontology, from this ontology we pick a concept T that will be our target. Then, we create, for each numeric data property, some fuzzy datatypes. We use as positive samples

during learning ontology individuals that can be proven belonging to target concept, negatives are picked in different ways depending on the open/closed world assumption. Then, using different algorithms, we try to learn fuzzy SubClassOf axioms. Formally the setting can be described as:

Definition 6. *Let $\mathcal{K}$ be a crisp* OWL 2 *ontology, let T be an atomic concept from $\mathcal{K}$. Axiom learning aims at finding axioms of the form: $C \sqsubseteq T$ where C is a legal $\mathcal{EL}(D)$ class expression.*

Assuming axioms obtained are $H = \{C_1 \sqsubseteq T, ..., C_n \sqsubseteq T\}$

- $\mathcal{K} \nvDash C_i \sqsubseteq T$ *with* $i = 1, ..., n$*;*
- $\mathcal{K} \cup H \nvDash \bot$*;*
- $\mathcal{K} \models a : T \Rightarrow \mathcal{K} \models a : C_i$ *for some* $i = 1, ..., n$ *and for each* $a \in \mathsf{Ind}$*.*

Chapter 4

DL-FOIL

Description **L**ogic **F**irst **O**rder **I**nductive **L**arning (*DL-FOIL*) is the first algorithm we are going to talk about. DL-FOIL derives from FOIL algorithm created by Quinlan [35]. This algorithm has been adapted to cope with description logics in [25, 26, 27].

DL-FOIL, like every other axiom learning algorithm in this work, tries to learn a good definition of an atomic concept. This definition is induced using a set of positive and negative samples.

This algorithm starts with a set of positive examples and a set of negative examples. A procedure, `learnOneAxiom`, is called. This procedure starts from top concept. A refinement operator [24] is applied. From the possible refinement of top concept, the best one (according to a performance measure) is picked. If the best refinement has a legal negative coverage it is learnt, otherwise refinement operator is applied to this concept expression until we get to a legal concept or no legal concept can be found. In the latter case the learning is finished and the algorithm outputs all the axioms learnt so far.

After the invocation of `learnOneAxiom`, if a legal concept expression has been found, all covered positive examples are removed from positive samples set.

The process is iterated until it is impossible to get a legal class expression from `learnOneAxiom` or until all positive samples are covered.

After this informal explanation, we are going to specify in more detail:

1. How refinement operator works;
2. Which performance evaluation has been used;
3. What legal negative coverage, and thus legal concept, means;
4. As we are learning fuzzy inclusion axioms, how degrees are assigned to learnt axioms.

Before starting the explanations, please note that the algorithm described is a greedy one, as it greedily look for a "good" refinement. A backtrack algorithm could reach better performance but it would have higher computation costs. However a best-k backtrack has been applied. It will be explained later how it works. For the moment being we just need to consider that the search is greedy.

4.1 Refinement Operator

In this section we are going to define a refinement operator as requested on point 1. Refinement operators can be considered function that associates to a concept expression a more general or more specific concept expression. A concept expression C is considered to be more general than a concept expression D if $D \sqsubseteq C$. If C is more general than D, we say that D is more specific than C. If a refinement outputs only more general concept expression from an input concept expression it is called *upward refinement operator*, denoted as $\rho_{\uparrow}$. When a refinement operator produces more specific concept expressions from a concept expression it is called *downward refinement operator*, denoted as $\rho_{\downarrow}$.

In this work downward refinement operators have been used almost always. When needed, an explicit distinction will be made. Otherwise, we will use ρ to denote downward refinement operators.

Let's now have a look at how the refinement operators work. As explained earlier, `learnOneAxiom` begins its search from top concept. Informally it has been said that learnt axioms should cover as less negative samples as possible. Obviously top concept does not respect this, as it covers all negative examples, if they exist. To reduce the number of negative examples covered we need more specific concept expressions. That's what downward refinement operators do. So `learnOneAxiom` exploits a downward refinement operator $\rho_{\downarrow}$ to make a concept more specific in such a way the "negative coverage" can be reduced. The refinement operator used is the same used in [25, 26, 27]. It works as follows:

Definition 7. *Let $\mathcal{K}$ be an ontology, $\mathcal{A}$ be the set of all atomic concept in $\mathcal{K}$, $\mathcal{R}$ the set of all object properties in $\mathcal{K}$, $\mathcal{S}$ the set of all data properties in $\mathcal{K}$ and $\mathcal{D}$ a set of (fuzzy) datatypes. ρ is a function that associates to a concept expression a set of concept expressions. Each element of the result set is a sub class of ρ's input.*

ρ as used in this work is defined in Table 4.1. It is possible to notice that ρ is defined only on a small subset of possible concept expressions. However

$$\rho(C) = \begin{cases} \mathcal{A} \cup \{\exists R.\top \mid R \in \mathcal{R}\} \cup \{\exists S.d \mid S \in \mathcal{S}, d \in \mathcal{D}\} & if \quad C = \top \\ \{A' \mid A' \in \mathcal{A}, A' \sqsubseteq A\} \cup \{A \sqcap A'' \mid A'' \in \rho(\top)\} & if \quad C = A \in \mathcal{A} \\ \{\exists R.D' \mid D' \in \rho(D)\} \cup \{(\exists R.D) \sqcap D'' \mid D'' \in \rho(\top)\} & if \quad C = \exists R.D \\ \{\exists (S.d) \sqcap D \mid D \in \rho(\top)\} & if \quad C = \exists S.d, S \in \mathcal{S}, d \in \mathcal{D} \\ \{C_1 \sqcap ... \sqcap C_i' \sqcap ... \sqcap C_n \mid i = 1, ..., n, C_i' \in \rho(C_i)\} & if \quad C = C_1 \sqcap ... \sqcap C_n \end{cases}$$

Table 4.1: Downward Refinement Operator.

it covers all $\mathcal{EL}(D)$ concept expressions used in this work.

Max Depth and Max Length. The refinement operator defined in Table 4.1 has an issue. It can lead to infinite application. There are two possible ways to have infinite generation:

- Applying refinement to $\exists R.\top$. In fact we have that a possible refinement of $\top$ is $\exists R.\top$. We would generate $\exists R.\exists R.\top$. It is clear that it is possible to apply this rule an infinite number of time;
- Applying refinement to a conjunction. In fact if we have an atomic concept into a conjunction, we can refine the conjunction adding a conjunct as, an atomic concept can be refined into a conjunction of that atomic with another. Applying again refinement we still would have the same atomic concept and so we could add another conjunct. And so on.

These two issues can be solved adding two threshold called *Max Depth* and *Max Length*.

Definition 8. *Let C be a concept expression, we define $length(C)$ as:*

$$length(C) = \begin{cases} 0 & if \quad C = A \ or \\ & \quad C = \exists S.d \ or \\ & \quad C = \exists R.D \\ length(C_1) + length(C_2) + 1 & if \quad C = C_1 \sqcap C_2 \end{cases}$$

We define $depth(C)$ as:

$$depth(C) = \begin{cases} 0 & if \quad C = A \ or \\ & \quad C = \exists S.d \ or \\ & \quad C = C_1 \sqcap C_2 \\ depth(D) + 1 & if \quad C = \exists R.D \end{cases}$$

We insert Max Depth and Max Length among DL-FOIL parameters. We will impose that $length(C) \leq maxLength$ and $depth(C) \leq maxDepth$ to avoid infinite refinement operations.

4.2 Performance Measures

We are now facing issue 2. This algorithm adopts a compound evaluation metric. First of all a confidence measure cf is used.

Definition 9. *Let $\mathcal{E}^+$ be the set of positive examples, $\mathcal{E}^-$ the set of negative individuals and $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$. We define a confidence measure over a concept expression C w.r.t. a target concept T as:*

$$cf(C) = \frac{\sum_{e \in \mathcal{E}^+} bed(\mathcal{K}, e : C)}{\sum_{e \in \mathcal{E}} bed(\mathcal{K}, e : C)}$$

cf defines how much a concept C can be considered "precise". A confidence value of 1 states that all examples are correctly classified by C, *i.e.*, C covers at least a positive and no negatives. A value of 0 means that C covers no positives, thus it is useless.

With cf at hand we define a *gain* measure.

Definition 10. *Let cf be the confidence measure defined in definition 9, we define the gain of concept expressions C w.r.t. D as:*

$$gain(C, D) = p \cdot (\log(cf(C)) - \log(cf(D)))$$

where $p = |\{e \in \mathcal{E}^+ \mid bed(\mathcal{K}, e : D) > 0$ and $bed(\mathcal{K}, e : C) > 0\}|$.

In our settlement we always consider the gain of C w.r.t. D where $C \sqsubseteq D$. In fact we use gain to evaluate the "best" refinement. All refinements with negative gain are ignored.

Thus *gain* is our measure to choose the best refinement of a certain concept expression. Moreover *gain* is a possible stop situation. In fact if we can't obtain an acceptable refinement, *i.e.*, a refinement whose *gain* is non negative, we reach a stop condition as the refined concept was not good (otherwise it would not have been refined) and no good refinement can be found. If it is the case, the algorithm stops and the axioms learnt until that moment are given back as the result. A similar situation happens when no refinements are available. We will see later when this can happen.

4.3 Negative Coverage

Negative coverage can be defined as follows:

Definition 11. *Let $\mathcal{E}^-$ be the set of negative examples, the negative coverage of a concept expression C is defined as:*

$$negCov(C) = \frac{\sum_{e \in \mathcal{E}^-} bed(\mathcal{K}, e : C)}{|\mathcal{E}^-|}$$

Informally negative coverage of a concept C can be defined as the percentage of negative samples covered. Note that *negCov* assumes that we are dealing with crisp target concept T otherwise $|\mathcal{E}^-|$ should be replaced by $\sum_{e \in \mathcal{E}^-} (\ominus bed(\mathcal{K}, e : T))$ where $\ominus$ is a fuzzy negation operator. If T is crisp the two are equivalent.

Clearly our goal is to keep negative coverage as low as possible. It is important to notice that if we are using OWA on a crisp ontology and we achieve a non zero negative coverage, the axioms learnt together with the original ontology originate an inconsistent ontology. In fact if $negCov(C) > 0$, then C covers at least one negative. So if we add axiom $C \sqsubseteq T$ to the original ontology $\mathcal{K}$ we obtain that for some $e \in \mathcal{E}^-$ we have $\mathcal{K} \cup \{C \sqsubseteq T\} \models e : T$. But as $e \in \mathcal{E}^-$ we have also that $\mathcal{K} \models e : \neg T$. Thus we have an inconsistency. However this may not be an inconsistency in a fuzzy ontology.

If we assume CWA we may still have an inconsistency, but not necessarily as if $e \in \mathcal{E}^-$ then $\mathcal{K} \models e : \neg T$ may not hold (*e.g.*, $\mathcal{K} \not\models e : T$ and $\mathcal{K} \not\models e : \neg T$)

In this work all of the algorithms give the possibility to define a negative coverage threshold stating that concept expressions with a higher negative coverage than the threshold should be avoided.

In DL-FOIL if we get a negative coverage higher than the threshold the search is continued until we get no admissible refinement or until we have a negative coverage low enough. If a crisp inconsistency has to be avoided, it is advisable to set the negative coverage to 0. In any case, DL-FOIL makes the *best effort* to lower the negative coverage as much as possible.

4.4 Axioms Degree

As last issue, let's have a look at how a degree is assigned to each axiom. Having defined a confidence measure in Definition 9, we can use that measure again. So we establish that an axiom has a degree equal to its confidence, *i.e.*, the degree of $C \sqsubseteq T$ is $cf(C)$.

Notice that if the negative coverage threshold is set to 0, the degree of $C \sqsubseteq T$ will be lower than 1 iff C involves a fuzzy concept, *i.e.*, a data restriction, as they are the only fuzzy concepts admitted.

4.5 The Algorithm

After having described the main ingredients of DL-FOIL, it is possible to describe the actual algorithm (see Algorithm 1). The external loop is a function called `learnSetOfAxioms` (Algorithm 1). This function has as parameters a knowledge base $\mathcal{K}$, a target atomic concept T from $\mathcal{K}$, a set of positive examples $\mathcal{E}^+$ and a set of negative examples $\mathcal{E}^-$. This function simply accumulate learnt axioms into result axiom set $\mathcal{H}$ and removes covered positive samples. This outer loop ends when all positive examples are covered or when `learnOneAxiom` returns `null` or $\top$. As can be noticed, `learnSetOfAxiom` has a quite simple structure. The working core of DL-FOIL is `learnOneAxiom`, *i.e.*, Algorithm 2.

Algorithm 1 DL-FOIL: External Loop

function `learnSetOfAxioms`($\mathcal{K}$, T, $\mathcal{E}^+$, $\mathcal{E}^-$)
  $\mathcal{H} \leftarrow \emptyset$;
  while $\mathcal{E}^+ \neq \emptyset$ **do**
    $\phi \leftarrow$ learnOneAxiom($\mathcal{K}$, T, $\mathcal{E}^+$, $\mathcal{E}^-$);
    if$(((\phi = \top)\ or\ (\phi = null)))$
      return $\mathcal{H}$;
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{\phi \sqsubseteq T\}$;
    $\mathcal{E}^+_\phi \leftarrow \{e \in \mathcal{E}^+ \mid bed(\mathcal{K}, e : \phi) > 0\}$;
    $\mathcal{E}^+ \leftarrow \mathcal{E}^+ \backslash \mathcal{E}^+_\phi$;
  return $\mathcal{H}$;
end

The function `learnOneAxiom`, as explained earlier, greedily search for the best acceptable concept expression in terms of gain. It starts from $\top$ and applies refinement until a good concept is found. So, the method `refine`, called in line 7 of Algorithm 2, is a method that returns all possible refinements of a concept expression as specified by refinement operator defined in Section 4.1.

Finally method `notLegalNegCov` is defined in Algorithm 3. It simply return true if ϕ has not a legal negative coverage as specified in Section 4.3. `threshold` is the threshold of acceptable negative coverage.

4.6 Backtrack Variant

A simple variant of DL-FOIL has been proposed and analysed. This simple variant uses a partial backtrack, called *Best-K Backtrack*. Best-K Backtrack saves the K best concept expression obtained from refinements. The stack is

Algorithm 2 DL-FOIL: Internal Loop

function `learnOneAxiom`($\mathcal{K}$,T,$\mathcal{E}^+$,$\mathcal{E}^-$)
  $\phi \leftarrow \top$;
  $\mathcal{E}^-_\phi \leftarrow \mathcal{E}^-$;
  while $((cf(\phi) < \theta)$ *or* $(\text{NOTLEGALNEGCOV}(\mathcal{K}, \phi, \mathcal{E}^-_\phi, \mathcal{E}^-)))$ **do**
    $\phi_{best} \leftarrow \phi$;
    $maxgain \leftarrow 0$;
    $\Phi \leftarrow \text{REFINE}(\phi)$;
    for all $\phi' \in \Phi$ **do**
      $gain \leftarrow \text{GAIN}(\phi', \phi)$;
      if $gain \geq maxgain$ **then**
        $maxgain \leftarrow gain$;
        $\phi_{best} \leftarrow \phi'$;
    if$(\phi_{best} = \phi)$
      return `null`;
    $\phi \leftarrow \phi_{best}$;
    $\mathcal{E}^-_\phi \leftarrow \{e \in \mathcal{E}^- \mid bed(\mathcal{K}, e : \phi) > 0\}$;
  return ϕ;
end

Algorithm 3 DL-FOIL: Negative Coverage Check

function `notLegalNegCov`($\mathcal{K}, \phi, \mathcal{E}^-_\phi$,$\mathcal{E}^-$)
  $negCov \leftarrow \sum_{e \in \mathcal{E}^-_\phi} bed(\mathcal{K}, e : \phi)$;
  return $\dfrac{negCov}{|\mathcal{E}^-|} \leq threshold$;
end

locally maintained during `learnOneAxiom`. Saving K possible candidates, if we get stuck we can recover a saved candidate and continue the search from it. Backtrack variant is described in Algorithm 4. Using backtrack we try to balance quality and complexity of the algorithm. `BESTK` function called in line 10, simply returns a set containing the K elements that have the highest gain w.r.t. ϕ. `NEXTBEST` function called in line 13 instead removes from stack the concept expression that has the highest gain w.r.t. ϕ. The removed concept is returned as result.

Algorithm 4 DL-FOIL: Internal Loop with Best-K Backtrack

1: **function** `learnOneAxiom`($\mathcal{K}$,T,$\mathcal{E}^+$,$\mathcal{E}^-$)
2: $\phi \leftarrow \top$;
3: $\mathcal{E}^-_\phi \leftarrow \mathcal{E}^-$;
4: $stack \leftarrow \emptyset$;
5: **while** $((cf(\phi) < \theta)$ *or* $(\text{NOTLEGALNEGCOV}(\mathcal{K}, \phi, \mathcal{E}^-_\phi, \mathcal{E}^-)))$ **do**
6: $\phi_{best} \leftarrow \phi$;
7: $maxgain \leftarrow 0$;
8: $\Phi \leftarrow \text{REFINE}(\phi)$;
9: $stack \leftarrow stack \cup \Phi$;
10: $stack \leftarrow \text{BESTK}(\phi, stack)$;
11: **if** $(\phi_{best} = \phi)$ **then**
12: **if** $stack \neq \emptyset$ **then**
13: $\phi_{best} \leftarrow \text{NEXTBEST}(\phi, stack)$;
14: **elsereturn** `null`;
15: $\phi \leftarrow \phi_{best}$;
16: $\mathcal{E}^-_\phi \leftarrow \{e \in \mathcal{E}^- \mid bed(\mathcal{K}, e : \phi) > 0\}$;
return ϕ;
17: **end**

Chapter 5

pFOIL

pFOIL is a probabilistic variant of DL-FOIL. It has two main differences from DL-FOIL:

- It uses a probabilistic measures to evaluate concept expressions;
- Instead of removing positive examples covered from the training set, once an axiom has been learnt, positive examples are left unchanged for all the invocations of `learnOneAxiom`. Moreover concept expressions are no more evaluated one at a time, but the whole set of learnt rules is evaluated as an ensemble.

5.1 Probability Estimation

Our first idea for pFOIL was to adapt naïve Bayes FOIL (*nFOIL*), proposed in [22]. nFOIL put together FOIL and the usage of probability together with naïve Bayes assumption to improve FOIL algorithm. Many attempts have been made to improve FOIL using probability. The original idea contained in [22] was that probability measure should be used during learning, as performance measure, and not after learning to prune results. We followed somewhat the same idea, but nFOIL was applied to logic programming and its goal was to learn conjunctive axioms. As nFOIL axioms were considered as conjunctions, naïve Bayes assumption was made to reduce complexity. In our case, we wanted to learn a set of axioms that should be considered as a disjunction. In fact if $C_1 \sqsubseteq T, ..., C_n \sqsubseteq T$ are the learnt axioms we should consider that $C_1, ..., C_n$ can be put together with a disjunction. In [22] instead the learnt axioms are considered as conjunction.

Let's have a look at how probability is computed and how it is used as score. Let's begin with a set containing only one axiom, $\mathcal{H} = \{C \sqsubseteq T\}$. We

define:

$$\begin{aligned} T^+ &= card(T) \\ C^+ &= card(C) \\ T^+ \sqcap C^+ &= card(T \sqcap C) \end{aligned}$$

where $card(\cdot)$ is a cardinality measure over the sample set $\mathcal{E}$.

Precision, Recall and $F_\beta - score$. In the following to compute the scores of a set of rules we used measures derived from precision, recall and $F_\beta - score$.

Precision, recall and $F_\beta - score$ are measures that evaluate goodness of classification.

Supposing we have a classifier C, a target T and a samples set $\mathcal{E}$. We say that:

1. An example e is classified as a *true positive* if it is classified as positive by C, *i.e.*, C predicts that it belongs to T, and e actually is in T;

2. An example e is classified as a *true negative* if it is classified as negative by C, *i.e.*, C predicts that it does not belong to T, and e actually is not in T;

3. An example e is classified as a *false positive* if it is classified as positive by C, *i.e.*, C predicts that it belongs to T, and instead e is not in T;

4. An example e is classified as a *false negative* if it is classified as negative by C, *i.e.*, C predicts that it does not belong to T, and instead e is in T;

The objective is to classify as many true positives and true negatives as possible. The other two cases are wrong predictions. Disposition is explained in Table 5.1.

Definition 12. *Let tp, fp, fn and tn be the number of true positive, false positive, false negative and true negative respectively. Let C be a classifier and T our target, we define:*

Precision: $prec(C,T) = \dfrac{tp}{tp + fp}$;

Recall: $rec(C,T) = \dfrac{tp}{tp + fn}$;

$F_\beta - score$**:** $F_\beta(C,T) = (1+\beta^2) \cdot \dfrac{prec(C,T) \cdot rec(C,T)}{(\beta^2 \cdot prec(C,T)) + rec(C,T)}$;

Classif. \ Target	Positive	Negative
Positive	True Positive (tp)	False Positive (fp)
Negative	False Negative (fn)	True Negative (tn)

Table 5.1: Classification Table.

The most used version of $F_\beta - score$ is the one with $\beta = 1$. However increasing β gives recall bigger relevance while decreasing it gives precision more relevance.

In this work the measures that we use are based on probability. So we define

$$P(C^+ \mid T^+) = \frac{C^+ \sqcap T^+}{T^+} \ ,$$

as the probability of correctly classifying a positive example, and

$$P(T^+ \mid C^+) = \frac{C^+ \sqcap T^+}{C^+} \ ,$$

as the probability that a sample classified as positive is actually positive.

The first probability measures the *recall* of C w.r.t. T while the second one measures the *precision* of C w.r.t. T. We can combine the two measures through a $F_\beta - score$ obtaining:

$$score(C) = (1 + \beta^2) \cdot \frac{P(T^+ \mid C^+) \cdot P(C^+ \mid T^+)}{(\beta^2 \cdot P(P(T^+ \mid C^+) + P(C^+ \mid T^+)}$$

5.2 Ensemble Evaluation

In DL-FOIL we have seen how to evaluate hypothesis having $\mathcal{H}$ with only one axiom. Here, instead, we want to evaluate the set of hypothesis as an *ensemble*.

While in DL-FOIL each iteration of `learnOneAxiom` have an evaluation of candidates on their own, in pFOIL we want `learnOneAxiom`'s evaluation to consider also previously learnt axioms.

Specifically, for $\mathcal{H} = \{C_1 \sqsubseteq T, ..., C_n \sqsubseteq T\}$, we define

$$\begin{aligned} T^+ &= card(T) \ , \\ \mathcal{H}^+ &= card(C_1 \sqcup ... \sqcup C_n) \ , \\ \mathcal{H}^+ \sqcap T^+ &= card((C_1 \sqcup ... \sqcup C_n) \sqcap T) \ . \end{aligned}$$

Notice that the definition of $\mathcal{H}^+ \sqcap T^+$ involves disjunction. Once we have at hand the degrees for $C_1, ..., C_n$ to compute the degree for $C_1 \sqcup ... \sqcup C_n$, we just need to combine them through an *s-norm*.

We have that our ensemble variant for precision, recall and $F_\beta - score$ is

$$\begin{aligned} prec(\mathcal{H},T) &= P(T^+ \mid \mathcal{H}^+) = \frac{\mathcal{H}^+, \sqcap T^+}{\mathcal{H}^+} \\ rec(\mathcal{H},T) &= P(\mathcal{H}^+ \mid T^+) = \frac{\mathcal{H}^+, \sqcap T^+}{T^+} \\ score_\beta(\mathcal{H},T) &= F_\beta(\mathcal{H},T) = (1+\beta) \cdot \frac{P(T^+ \mid \mathcal{H}^+) \cdot P(\mathcal{H}^+ \mid T^+)}{(\beta^2 \cdot P(T^+ \mid \mathcal{H}^+)) + P(\mathcal{H}^+ \mid T^+)} \end{aligned}$$

In summary, to evaluate the performance of a concept expression ϕ w.r.t. axioms $\mathcal{H} = \{C_1 \sqsubseteq T, ..., C_n \sqsubseteq T\}$ that have been already learnt, we shall compute $score(\mathcal{H} \cup \{\phi \sqsubseteq T\}, T)$ as our scoring function.

5.3 Stop Criterion

In DL-FOIL we stopped the algorithm when positive samples were all covered or when it became impossible to find a concept with a legal negative coverage. In pFOIL, instead, we impose that no axioms are learnt unless they improve the ensemble score. So if adding a new axiom the score of ensemble does not vary (or decreases) the axiom is not learnt.

Moreover, we use a threshold and stop a soon as the score improvement is below the threshold.

Once an axiom has been learnt, the ensemble performance is evaluated through the $F_\beta - score$. However, this evaluation is a different evaluation w.r.t. the evaluation made for candidates concept expressions. It would be useful to define different a value for β to be used during concept evaluation and another one to be used during ensemble evaluation. In this work the two are considered independent and so may have different values. In the following, the β used during concept evaluation will be denoted as β_1 while the value used during ensemble evaluation will be denoted as β_2.

5.4 The Algorithm

The pFOIL algorithm is defined in Algorithm 5. As for DL-FOIL we have to define both `learnSetOfAxioms` method and `learnOneAxiom` method.

`learnSetOfAxioms`, Algorithm 5, is very similar to the DL-FOIL one. But, a first difference is in line 5: the stop criterion is no more on positive samples size but on score increment. Notice that the first score is evaluated w.r.t. $\mathcal{H} = \{\top\}$. This means that at the beginning we consider a naïve classifier that classifies all individuals as belonging to target concept. The

Algorithm 5 pFOIL: learnSetOfAxioms

1: **function** `learnSetOfAxioms`($\mathcal{K}$, T, $\mathcal{E}^+$, $\mathcal{E}^-$, θ, β_1, β_2)
2: $\mathcal{H} \leftarrow \emptyset$;
3: $oldScore \leftarrow 0$;
4: $newScore \leftarrow score_{\beta_2}(\{\top \sqsubseteq T\}, T)$;
5: **while** $newScore - oldScore > \theta$ **do**
6: $\phi \leftarrow$ learnOneAxiom($\mathcal{K}$, T, $\mathcal{H}$, $\mathcal{E}^+$, $\mathcal{E}^-$, β_1);
7: **if**$(((\phi = \top)$ *or* $(\phi == null)))$
8: **return** $\mathcal{H}$;
9: $\mathcal{H} \leftarrow \mathcal{H} \cup \{\phi \sqsubseteq T\}$;
10: $oldScore \leftarrow newscore$
11: $newScore \leftarrow score_{\beta_2}(\mathcal{H}, T)$;
return $\mathcal{H}$;
12: **end**

explicit form of first score is:

$$\begin{aligned} prec(\{\top \sqsubseteq T\}, T) &= \tfrac{|\mathcal{E}^+|}{|\mathcal{E}|} \ , \\ rec(\{\top \sqsubseteq T\}, T) &= 1 \ , \\ score(\{\top \sqsubseteq T\}, T) &= (1+\beta_2) \cdot \tfrac{\frac{|\mathcal{E}^+|}{|\mathcal{E}|}}{(\beta_2^2 \cdot \frac{|\mathcal{E}^+|}{|\mathcal{E}|})+1} \ . \end{aligned}$$

The other significant difference stands in the fact that positive samples removal has been eliminated. It has been replaced by a score update.

Algorithm 6 instead is the `learnOneAxiom` for pFOIL. In this case we have few significant differences. We have changed the score evaluation. More important we do not have $\mathcal{E}^-_\phi$ and we use `noteLegalNegCov` method (see Algorithm 7).

Note that in Algorithm 7 we have replaced negative coverage of ϕ with its cardinality. Notice that in Algorithm 3 we explicitly used σ-count. Moreover cardinality of target is computed and replaced $|\mathcal{E}^-|$. [1]

5.5 Backtrack

Like for DL-FOIL in pFOIL backtrack is possible and reported in Algorithm 8[2].

[1] In this case, we increase usability and admit T to be fuzzy. However if target is crisp, the cardinality works exactly as $|\mathcal{E}^-|$.

[2] Methods `BESTK` and `NEXTBEST` are exactly the same as for DL-FOIL.

Algorithm 6 pFOIL: learnOneAxiom

```
1: function learnOneAxiom(𝒦, T, ℋ, ℰ⁺, ℰ⁻, β₁)
2:     φ ← ⊤;
3:     while notLegalNegCov(𝒦, φ, ℰ⁻) do
4:         φ_best ← φ;
5:         maxscore ← score_β1(ℋ ∪ {φ_best ⊑ T}, T);
6:         Φ ← refine(φ);
7:         for all φ′ ∈ Φ do
8:             score ← score_β1(ℋ ∪ {φ′ ⊑ T}, T);
9:             if score ≥ maxscore then
10:                maxscore ← score;
11:                φ_best ← φ′;
12:        if(φ_best = φ)
13:            return null;
14:        φ ← φ_best;
       return φ;
15: end
```

Algorithm 7 pFOIL: Negative Coverage Check

```
1: function notLegalNegCov(𝒦, φ, ℰ⁻)
2:     negCov ← card_ℰ⁻(φ);
3:     negSize ← card_ℰ⁻(T);
4:     return negCov / negSize ≤ threshold;
5: end
```

Algorithm 8 pFOIL: learnOneAxiom with Best-K Backtrack

```
1: function learnOneAxiom(𝒦, T, ℋ, ℰ⁺, ℰ⁻, β₁)
2:     φ ← ⊤;
3:     stack ← ∅;
4:     while notLegalNegCov(𝒦, φ, ℰ⁻) do
5:         φ_best ← φ;
6:         maxscore ← score_β1(ℋ ∪ {φ_best ⊑ T}, T);
7:         Φ ← refine(φ);
8:         stack ← stack ∪ Φ;
9:         stack ←bestk(φ, stack);
10:        for all φ′ ∈ Φ do
11:            score ← score_β1(ℋ ∪ {φ′ ⊑ T}, T);
12:            if score ≥ maxscore then
13:                maxscore ← score;
14:                φ_best ← φ′;
15:        if φ_best = φ then
16:            if stack ≠ ∅ then
17:                φ_best ←nextBest(φ, stack);
18:            elsereturn null;
19:        φ ← φ_best;
       return φ;
20: end
```

Chapter 6

Hybrid Learning

In this chapter we present *Hybrid Learning*. It is an application of genetic programming to DLs. The original idea was proposed by Lehamann in [23]. In this work small changes have been made.

The basic idea of genetic programming is to emulate evolution of a population of hypothesis in such a way the "fittest" hypothesis would survive and hand down his good genes. Usually genetic programming is applied on sets of elements that can be represented as trees. *OWL 2*, and DL, concept expressions can easily be represented as trees. In fact, as they are obtained from a grammar, we can represent a concept expression with its parse tree.

Once we can have such a tree like representation of elements, we have to define a crossover operator. This operator takes two concepts and combines them into one or more elements that are called the offspring. Due to the tree representation, an easy way of achieving this is to split each of the two trees into two subtrees. The obtained subtrees get merged to originate an offspring.

Example 2. *Suppose we have two concept expression* $C = \exists R_1.(A_1 \sqcap A_2)$ *and* $D = A_3 \sqcap \exists R_2.\top$. *We can represent them as trees, Figure 6.1(a), and then define a split point as Figure 6.1(b). Once we have a split point we can apply crossover and obtain offspring as in Figure 6.1(c).*

This operation is a typical cross over operation for genetic programming. Offspring is added to population and during the successive iteration, the selection process will select population randomly following a distribution that gives to each individual (concept expression) a probability to survive that is proportional to its evaluation.

It is important to note that crossover operator does not consider the ontology as background knowledge. Due to this in [23] beside crossover also a refinement operator was used. [23] proposed to use refine on a general

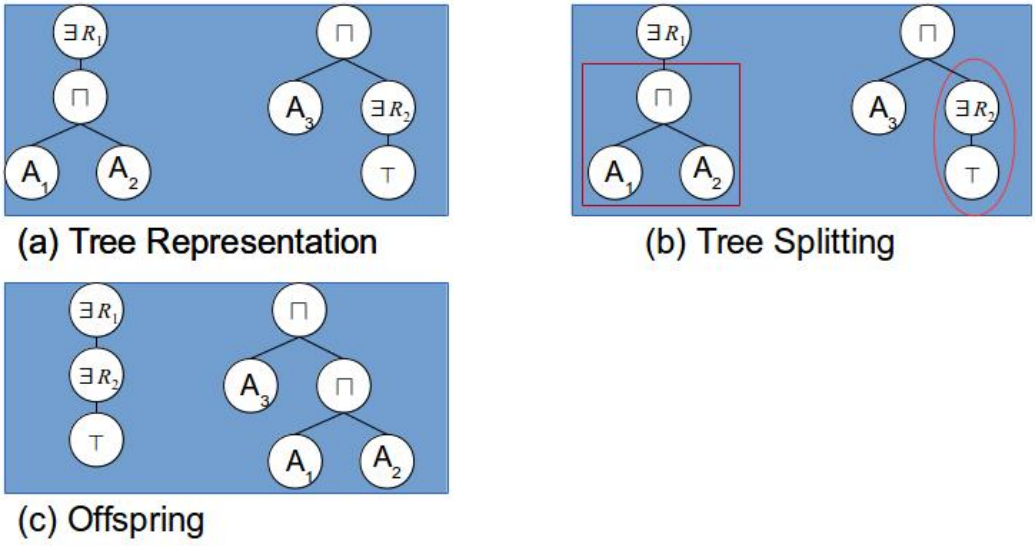

Figure 6.1: Crossover Example

description logic and to adopt both a downward and an upward refinement operator. In our case we adopt only the downward refinement operator, as defined in Section 4.1.

6.1 The Algorithm

Algorithm 9 Hybrid Learning Algorithm

```
1: function runHybridLearning(K, N, T, E+, E−)
2:     population ←InitializePopulation(K);
3:     for i = 1, ..., N do
4:         distribution ←ComputeDistribution(population, T, E+, E−);
5:         selectedPopulation ←Selection(population, distribution);
6:         reproducedPopulation ←Reproduction(selectedPopulation);
7:         mutatedPopulation ←Mutation(reproducedPopulation);
8:         population ← mutatedPopulation;
       return BestIndividual(population);
```

Algorithm 9 depicts how hybrid learning works. First of all, a population gets initialized. In our setting, the initial population is composed of all the

useful $\top$ refinements, *i.e.*, the concepts that cover at least a positive sample. Then N iterations are made. At each iteration we generate a distribution on the current population that gives higher probability to individuals that have a higher score. Based on this distribution, we operate a selection obtaining the selected population. These selected individuals get involved in reproduction. After reproduction we have a population containing both parents and offspring. On reproduced population we randomly apply mutation. Mutation is an operation that with very low probability modifies an individual, ignoring score and background knowledge. Mutated population is now considered as the current population. Finally Hybrid learning returns the best individual, *i.e.*, the one with highest score. Score function is left abstract, *i.e.*, every score function can be used.

It is important to note that target class and samples are useful only during the definition of the distribution (line 4). In fact samples and target are only used to get evaluation of concept expressions contained in the current population.

6.1.1 Computing distribution and selection

The method called to compute distribution (line 4) computes a distribution in such a way that concepts with higher score will receive a higher probability.

Selection is performed randomly following the computed distribution. To obtain a selection,a *roulette wheel* selection is performed. This kind of selection chooses randomly a number in $[0, 1]$, called *wheel outcome*. After that, an accumulator is initialized to zero and a concept is picked up from population: if it has a probability higher than the wheel outcome then it is selected otherwise its probability is added to the accumulator. A second concept is chosen, and this time we take the sum of the probability and the value of the accumulator. If the result is bigger than the wheel outcome, the concept is chosen otherwise the accumulator gets the computed sum. The process is iterated until a concept is chosen.

The whole procedure is iterated until we have selected enough concepts to survive, *i.e.*, until we have selected `ps`$\cdot$`popSize` concepts (the value is rounded up). Where `ps` is the faction of population that survives the selection, while `popSize` is the maximum size that the population can achieve.

6.1.2 Reproduction

Algorithm 10 illustrates the reproduction procedure. At line 8 and 10 we randomly pick an index to choose which type of reproduction will be chosen. The choice is among crossover, refine and forward. The latter simply adds an

Algorithm 10 Hybrid Learning: Reproduction

```
1: function REPRODUCTION(selectedPopulation)
2:     population ← selectedPopulation;
3:     reproducedPopulation ← ∅;
4:     while not ISEMPTY(population) do
5:         algType ← 0;
6:         offspring ← ∅;
7:         if SIZE(population)≥ 2 then
8:             algType ←RANDOM(0,1,2);
9:         else
10:            repType ←RANDOM(1,2);
11:        if repType = 0 then
12:            C1 ←RANDPICKANDREMOVEFROM(population);
13:            C2 ←RANDPICKANDREMOVEFROM(population);
14:            offspring ←CROSSOVER(C1, C2);
15:        else if repType = 1 then
16:            C1 ←RANDPICKANDREMOVEFROM(population);
17:            offspring ←REFINE(C1)
18:        else
19:            C1 ←RANDPICKANDREMOVEFROM(population);
20:            offspring ← {C1};
21:        ADD(reproducedPopulation, offspring);
       return reproducedPopulation;
```

individual to next population. In this algorithm the three reproductions are uniformly distributed. In our setting the distribution can be defined by user. In such a way it is possible to choose with which probability crossover and refine will be used. However, crossover should never get a probability of zero. In fact an uninformed crossover could avoid to get stuck on local minima.

If the refine method is selected, a concept gets removed from population. It is used as argument to refine. `refine` method returns a set containing all the useful refinements, *i.e.*, those who cover at least a positive. The result of refine also contains the refined concept.

If crossover is selected, two individuals are removed from population and they get crossed. The result returned is a set containing the two individuals picked and the offspring of them. Crossover is performed as explained previously (see Figure 6.1). Crossover can only be applied when the population has at least two individuals. This is the reason behind distinction from line 8 and line 10 of Algorithm 10.

Finally if forward is picked a concept is removed from population and it is put into reproduced population.

6.1.3 Mutation

Mutation, called on line 8 of Algorithm 9, is a procedure that iterates over a population. For each individual a mutation is made with very low probability. So every individual of a population can be mutated, but with very low probability. This operation is used to emulate natural evolution. In fact also in nature sometimes mutations occur introducing new characteristics, sometimes very useful ones. In genetic programming mutation can avoid local minima. In this work mutations are made through conjunction. Given a concept expression C, if C is a conjunction then we have two possibilities either a conjunct is added or one is replaced. If the concept expression is not a conjunction, we simply add a conjunct. Every new concept expression involved in mutation is a refinement of $\top$. Both the type of mutation end the $\top$ refinement used to mutate are chosen randomly. The mutation is made without considering background ontology.

Example 3. *Suppose that we have*

$$\rho(\top) = \{A_1, A_2, A_3, \exists R_1.\top, \exists R_2.\top, \exists S.d_1, \exists S.d_2\}$$

and two concept expressions $C_1 = A_3 \sqcap \exists S.d_2$ *and* $C_2 = \exists R_2.A_1$. *Then we have:*

- $mutation(C_1) \supset \{(\exists R_1.\top) \sqcap (\exists S.d_2), A_3 \sqcap (\exists S.d_2) \sqcap A_2\}$;

- $mutation(C_2) \supset \{(\exists R_2.A_1) \sqcap (\exists S.d_2), (\exists R_2.A_1) \sqcap A_1\}$;

where $muation(C)$ *is the set of all possible mutations of* C.

6.2 Usage of Hybrid Learning

Hybrid learning iterates for a certain number of times. When it finishes, the final population will be composed of different concept expressions. In [23], hybrid learning is executed on $\mathcal{ALC}$ concept expressions. This means that disjunctions are admitted. If $C_1 \sqcup C_2 \sqcup C_3 \sqsubseteq T$ is learnt in crisp $\mathcal{ALC}$, it can be considered as having learnt $C_1 \sqsubseteq T$, $C_2 \sqsubseteq T$ and $C_3 \sqsubseteq T$. In our setting instead this is not possible and we can have two possibilities:

1. Use genetic programming on set of concept expressions;
2. Use hybrid learning to learn *one* rule, modify the problem and run hybrid learning again.

In the first case we have to ignore ontology information and give a completely different formulation of the problem. So, we choose to use the latter approach.

So far we have already defined two algorithms that behave in this way. DL-FOIL learns one axiom, then modifies the set of positive examples and runs again the same procedure. pFOIL instead learns one axiom, modifies the scoring function considering the ensemble of learnt axioms and then runs again the same procedure. Therefore, to involve hybrid learning all we need to do is to use hybrid learning to learn the single axiom and let DL-FOIL and pFOIL modify the problem as they usually do.

6.3 gFOIL

gFOIL is a simple modification of DL-FOIL. In this algorithm `learnOneAxiom` simply call hybrid learning. `learnSetOfAxioms` is unchanged. The only thing that should be specified is how to evaluate concept expressions. We adopt as score the cf function of DL-FOIL (Definition 9). The problem is changed at every invocation of hybrid learning because `learnSetOfAxiom` modifies the set of positive examples and so the evaluation is different.

6.4 pgFOIL

pgFOIL combines both probabilistic measures and hybrid learning. It is obtained by modifying pFOIL. `learnSetOfAxioms` is unchanged while

`learnOneAxiom` simply calls hybrid learning. As above we need to define an evaluation function. The evaluation was made with the probability measure defined in Section 5.1. The problem is changed at each iteration because the evaluation function is modified adding the axiom previously learnt. This means that during the first iteration each candidate is evaluated on its own. During the second iteration instead, each candidate is evaluated as an ensemble with the axiom learnt during the first iteration. Successively each candidate is evaluated as an ensemble with all the axioms learnt so far.

Chapter 7

gAdaBoost

gAdaBoost is an algorithm that differs for many aspects from others. The main idea is the one of AdaBoost. In general, we learn some weak learner and combine them at the end. However the way our algorithm works is quite different from the original idea.

The original *Adaptive Boost* (AdaBoost) algorithm was proposed by Freund and Schapire in [11]. The main idea was to acquire a certain number of weak learners and then combine them in a weighted sum to obtain a strong learner. The original formulation worked with boolean target and boolean classifier. The peculiarity of this algorithm is that it considers weighted samples. At the beginning samples are uniformly distributed. After a weak learner is acquired, samples weights get updated in such a way that samples wrongly classified have higher weight w.r.t. correctly classified ones.

Algorithm 11 General AdaBoost algorithm

Input: sample $\mathcal{S} = \{\langle \mathbf{x_i}, y_i \rangle \mid \mathbf{x_i} \in \mathcal{X}, y_i \in \{0, 1\}\}$;

1: $w_1 \leftarrow u$;
2: **for** $t = 1, ..., N$ **do**
3: $\quad h_t \leftarrow WL(\mathcal{S}, w_t)$;
4: $\quad$ FIND α_t;
5: $\quad$ **for all** $1 \leq i \leq m$ **do**
6: $\quad\quad w_{t+1,i} \leftarrow w_{t,i} \times exp(-\alpha_t y_i h_t(\mathbf{x}_i))$;
7: $\quad$ NORMALIZE(w_{t+1});

Output: $H_N(x) = \sum_{t=1}^{N} \alpha_t h_t(\mathbf{x})$;

Algorithm 11 depicts a general version of AdaBoost. Many statements are deliberately left abstract. Line 4 asks to find a certain value α_t that will be

used to give a weight to t-th classifier. α will be set to a value that optimizes overall classification.

In line 3 a method to obtain a weak learner is called. This method will return a weak learner as a classifying function h_t.

Line 6 is used to update the weight of training examples. So w_t will be a vector containing a weight $w_{t,i}$ for each of the m samples (so $1 \leq i \leq m$). At the beginning, the weights are uniform. After that, they will be update as described in Line 6.

After each weights update, the new weights must be normalized in such a way they sum up to 1. This is done on line 7.

At the end of the algorithm, weak learners are put together to form the final learner. Line 7 describes how how the strong learner is obtained. As underlined before, the classification outputs the sign of final classifier.

Let us point out that this formulation does not work in our case because we don't have a boolean (crisp) classifier. In fact, dealing with fuzzy axioms, a classifier does not output a value in $\{0, 1\}$ but a value in $[0, 1]$. So instead of using the original idea of AdaBoost we use $\mathbb{R}$eal AdaBoost proposed by Nock and Nielsen in [30]. In this version the target is still crisp but classifiers outputs a real value; the classification is intended to classify as positive those samples whose classification value is bigger than zero and the others as negative.

7.1 $\mathbb{R}$eal AdaBoost

Let's see informally how $\mathbb{R}$eal AdaBoost works.

$\mathbb{R}$eal AdaBoost is as AdaBoost, but allows to deal with classifiers that outputs a real value instead of a crisp one. General functioning is maintained. Weight update and goal function are modified.

Specifically the changes involve only α, weights update and obviously the score function that is considered as weak learner's learning function. Indeed, we have that the:

- new weight update rule is

$$w_{t+1,i} \leftarrow w_{t,i} \times \left(\frac{1 - \frac{(\mu_t y_i h_t(\mathbf{x_i}))}{h_t^\star}}{1 - \mu_t^2} \right) \ ;$$

- new α_t value is computed as:

$$\frac{1}{2h_t^\star} \log \frac{1 + \mu_t}{1 - \mu_t} \ ;$$

- weak learners scoring function is:

$$\mu_t = \frac{1}{h_t^\star} \sum_{i=1}^{m} w_{t,i} y_i h_t \mathbf{x_i} \in [-1, +1] \ ;$$

where $h_t^\star = max_{1 \leq i \leq m} |h_t(\mathbf{x_i})|$. It should be noted that while, in general, in AdaBoost the function used to train weak learners was an error function, *i.e.*, it had to be minimized; in this setting μ_t is a score function, *i.e.*, it has to be maximized. This perfectly fits our setting as we have always used scoring functions.

$\mathbb{R}$eal AdaBoost has been proven in [30] to be able to build a strong learner from weak learners.

7.2 gAdaBoost

In general, $\mathbb{R}$eal AdaBoost and AdaBoost aggregate weak learners to obtain a strong learner. We, instead, learn at each iteration a single axiom.

gAdaBoost uses $\mathbb{R}$eal AdaBoost weights and score functions. So it is entirely specified unless how weak learners are obtained. We want to learn one axiom each time a weak learner has to be obtained. Among the previous methods to learn one axiom (DL-FOIL, pFOIL `learnOneAxiom` and hybrid learning), hybrid learning has be chosen as the method to learn the axiom at each iteration. The principal reason behind this choice stands in the fact that hybrid learning is the most customizable method among the three. In fact it is quite easy to let hybrid learning consider samples weights. It suffices to use a score function that not only considers coverage, but also individual weights.

Hybrid learning can be considered as a stand alone algorithm and does not exploit a specific scoring function and, thus, the $\mathbb{R}$eal AdaBoost scoring function can be used. Notice that the DL-FOIL `learnOneAxiom` works only coupled with `learnsetOfAxioms`. Similarly, pFOIL `learnOneAxiom` is not usable as it evaluates an ensemble of rules, while in our case we need a method that evaluates a concept at a time.

To do so, first of all we have to obtain a sample set with correct structure. Remember that our samples set must have the form $\mathcal{S} = \{\langle x_i, y_i \rangle \mid x_i \in \mathcal{X}, y_i \in \{-1, 1\}\}$. So we build our sample set with $\mathcal{X}$ being the set of individuals and $y_i = 2 \times bed(\mathcal{K}, x_i : T) - 1$. Notice that if the target concept is crisp the setting perfectly fits $\mathbb{R}$eal AdaBoost requirements.

Having modified the target semantics, we have to correct two more things. (1) The scoring function values are in $[-1, 1]$. But hybrid learning cannot deal

with negative scores. To patch this, we filtered μ_t values. If a value is scored with a value $v \leq 0$ then its score is set to zero. With this filter we guarantee to remove negative scores, *i.e.*, a concept expression will have a zero score score only if it covers more negatives than positives. It is important to point out that value of μ_t is filtered only during evaluation in hybrid learning. In every other case it is left unchanged.

Once axioms have been induced, we need a method to classify the new elements. Usually AdaBoost and $\mathbb{R}$eal AdaBoost use weighted sum. In our case, we can use the weighted sum as it can be expressed as a fuzzy concept of Fuzzy *OWL 2*.

However, note that AdaBoost and $\mathbb{R}$eal AdaBoost tend to acquire learners that cover more or less the same set of examples, *i.e.*, they acquire rules that are not mutually exclusive, while in our setting, the induced rules tend to be mutually exclusive w.r.t. coverage. Therefore, we try two different versions: one in which we use the weighted sum as aggregation function, and one in which we use the max only as aggregation function.

Part III

Evaluation

Chapter 8

Evaluation

An evaluation of the proposed algorithms has been carried on. Evaluation has been made on some ontologies. Our evaluation aims at comparing the proposed algorithms. Evaluation has been carried out with two different approaches. For some ontologies, k-fold cross validation has been run and comparisons are made on squared error, precision, recall and $F_1 - score$. For the other ontologies tests have been made in order to find whether or not the algorithm is able to find "reasonable" axioms.

8.1 The Ontologies

Below Table 8.1 reports the ontologies used and some of their statistics.

8.2 Results

The ontologies have been tested through k-fold cross validation or by a "goal oriented" method, *i.e.*, with the goal of learning a correct axiom. For illustrative purposes, we report also the induced axioms (in case of the k-fold cross validation test, we provide the set of induced axioms of one of the k runs).

8.2.1 The Father Ontology

This ontology has been tested following the goal oriented approach and assuming the CWA. The target was concept Father, max length was 2 and max depth 1. This ontology has produced positive results for all the algorithms. All of them have induced at least the rule:

```
male and (hasChild some Thing) SubClassOf Father
```

This rule is indeed correct. Some algorithms also induced

Ontology	# logical axioms	# classes	# object properties	# data properties	# individuals	DL expressivity
Father	14	3	1	0	6	$\mathcal{ALC}$
Hotel2	749	89	3	1	88	$\mathcal{ALCHOF}(D)$
Moral	4869	46	0	0	202	$\mathcal{ALC}$
Trains2	344	32	5	0	50	$\mathcal{ALCO}$

Table 8.1: Ontologies Used.

`male and (hasChild some male) SubClassOf Father`

This rule on its own is too specific but together with the first one it becomes redundant and so does not affect classification quality.

8.2.2 The Hotel2 Ontology

This ontology has been tested through k-fold cross validation and using Good_Hotel as target concept. Max length was set to 3 and max depth to 2. Data property hasPrice has been provided with 5 fuzzy datatypes hasPrice_veryLow, hasPrice_low, hasPrice_medium, hasPrice_high and hasPrice_veryHigh. These datatypes were obtained through triangular same width discretization. hasPrice_veryLow has been defined as a left shoulder $ls(45, 63.2)$, hasPrice_low a triangular function $tri(45, 63.2, 81.4)$, hasPrice_medium with a triangular function $tri(63.2, 81.4, 99.6)$, hasPrice_high with a triangular function $tri(81.4, 99.6, 117.8)$ and hasPrice_veryHigh with a right shoulder $rs(99.6, 117.8)$. The results of a 5-fold cross validation have been reported in Table 8.2.

As we can see from statistics, the most performing algorithms is pFOIL. It got the best squared error and the best $F_1 - score$. pFOIL reach full precision classification. It means that for each of the 5 runs pFOIL has reached full precision, *i.e.*, it never gets false positive. This is indeed an important result as precision is very important in our setting. In fact, it should be remembered that if an axiom classifies a negative sample as positive, when such axiom is added to the original ontology, the ontology obtained is inconsistent.

Another important aspect that should be noticed is related to precision

Algorithm	Squared Error	Precision	Recall	$F_1 - score$	AVG Time
Gredy FOIL	0.1759	0.8326	0.4731	0.5884	1336 ms
Best-5 Backtrack FOIL	0.1559	0.9	0.4731	0.6073	860 ms
pFOIL	0.1054	1	0.5956	0.7252	1689 ms
gFOIL	0.1748	0.7985	0.5627	0.6293	14956 ms
pgFOIL	0.2135	0.4917	0.75	0.5536	5885
gAdaBoost Weighted	0.2297	0.6554	0.1115	0.1837	21756 ms
gAdaBoost Thresholded	0.2826	0.6167	0.3956	0.4010	21756 ms

Table 8.2: Hotel2 5-fold cross validation average results.

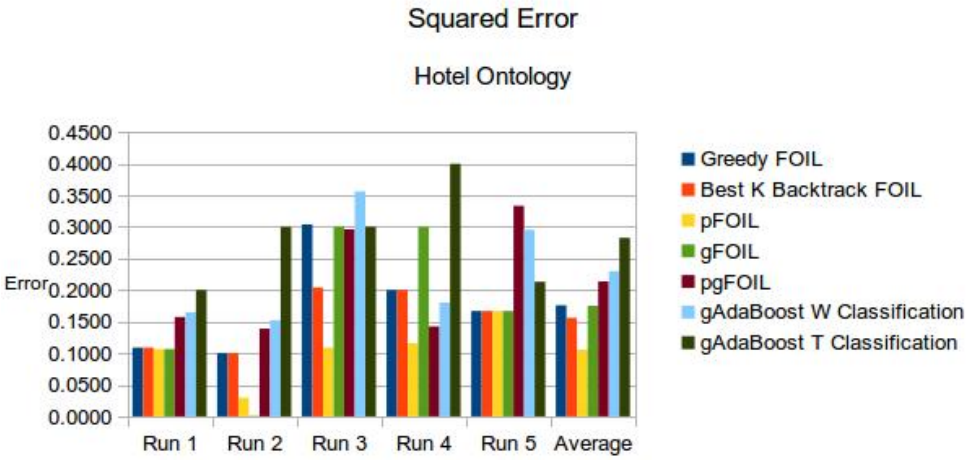

Figure 8.1: Hotel average squared error for the 5 runs of 5-fold cross validation.

and recall. Algorithms that involve hybrid learning suffer from low precision. Algorithms that do not exploit hybrid learning suffers from low recall but can reach high precision. This means that hybrid learning tries to find particular concept expressions reducing precision. Whenever high precision is needed, algorithms that do not exploit hybrid learning seem to perform better. Hybrid learning algorithms seem to suffer from overfitting.

It is important to notice that Weighted gAdaBoost seems to perform very badly. However it can reach a high precision. This happens because this kind of classification adopted for gAdaBoost is an extremely safe one.

Figures 8.1 - 8.4 depicts the scores achieved by the algorithms for each run.

Following we report the axioms learnt during the first run.

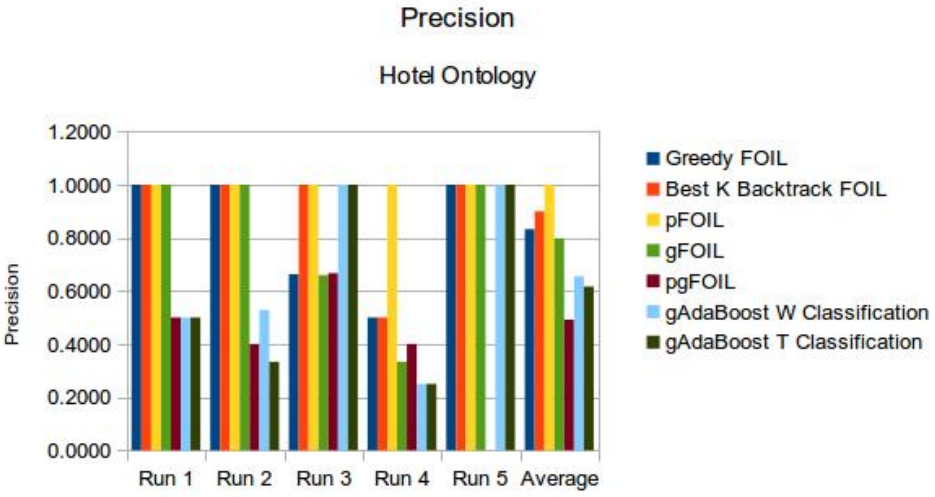

Figure 8.2: Hotel average precision for the 5 runs of 5-fold cross validation.

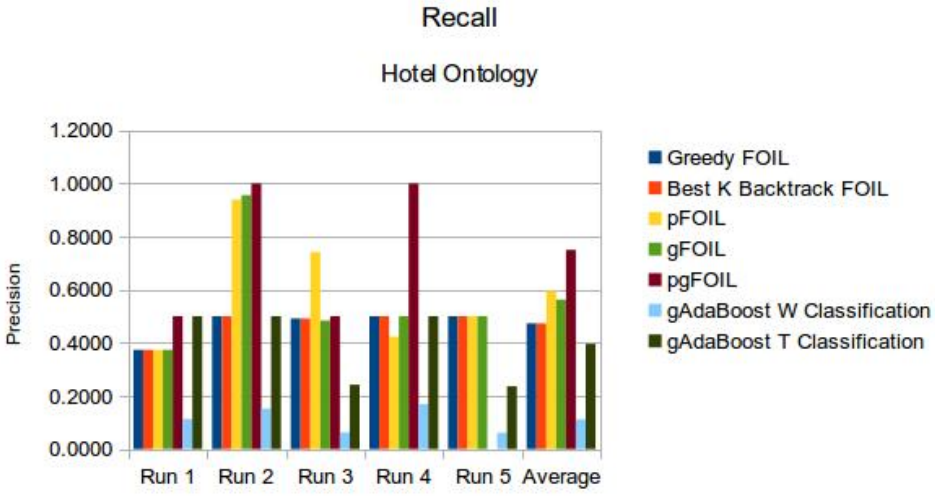

Figure 8.3: Hotel average recall for the 5 runs of 5-fold cross validation.

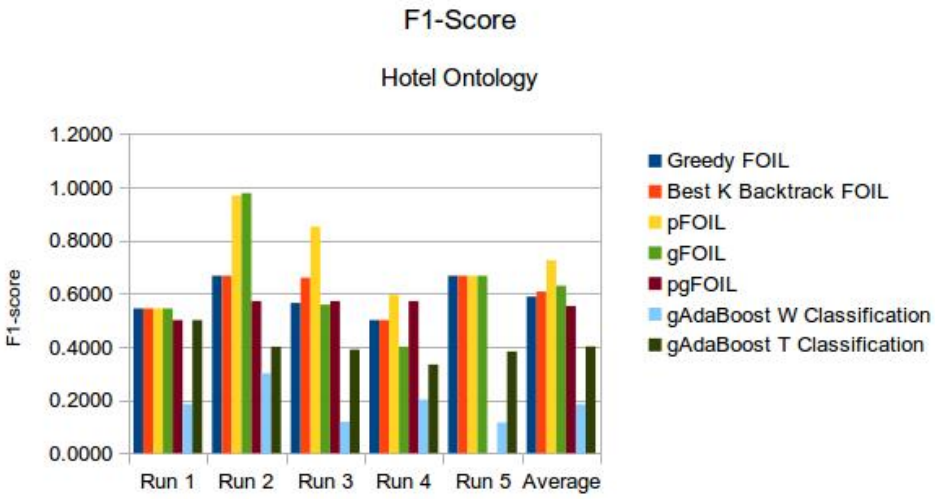

Figure 8.4: Hotel average $F_1 - score$ for the 5 runs of 5-fold cross validation.

Greedy FOIL

```
Site_Near_Civic SubClassOf Good_Hotel
 Hostel SubClassOf Good_Hotel
 Hotel_4_Stars and (hasPrice some hasPrice_veryHigh)
    SubClassOf Good_Hotel
 Hotel_1_Star SubClassOf Good_Hotel
 Bed_and_Breakfast and (hasPrice some hasPrice_high)
    SubClassOf Good_Hotel
 Hotel_4_Stars and (hasAmenity some Babysitting)
    SubClassOf Good_Hotel
```

Best-5 Backtrack FOIL

```
Site_Near_Civic SubClassOf Good_Hotel
 Hostel SubClassOf Good_Hotel
 Hotel_4_Stars and (hasPrice some hasPrice_veryHigh)
    SubClassOf Good_Hotel
 Hotel_1_Star SubClassOf Good_Hotel
 Bed_and_Breakfast and (hasPrice some hasPrice_high)
    SubClassOf Good_Hotel
 Hotel_4_Stars and (hasAmenity some Swimming_Pool)
    SubClassOf Good_Hotel
```

pFOIL

```
  Bed_and_Breakfast and (hasPrice some hasPrice_high)
      SubClassOf tempConcept_0
   hasPrice some hasPrice_veryHigh SubClassOf tempConcept_1
   Hostel SubClassOf tempConcept_2
   Hotel_5_Stars and Site_Near_University
      SubClassOf tempConcept_3
   Site_Near_Bridge and (hasPrice some hasPrice_high)
      SubClassOf tempConcept_4
   (tempConcept_0 or tempConcept_1 or tempConcept_2 or
      tempConcept_3 or tempConcept_4) SubClassOf Good_Hotel
```

gFOIL

```
  Hotel_3_Stars and Site_Near_University SubClassOf Good_Hotel
   Hotel_1_Star and (hasAmenity some (Beach-Volley and (hasAmenity
some Sub)))) SubClassOf Good_Hotel
   Hotel_5_Stars and Site_Near_University
      SubClassOf Good_Hotel
   Hotel_4_Stars and Site_Near_Civic SubClassOf Good_Hotel
   Site_Near_University and (hasAmenity some Cradle)
      SubClassOf Good_Hotel
   Hostel and (hasPrice some hasPrice_veryLow)
      SubClassOf Good_Hotel
   Hostel and (hasAmenity some Parking) SubClassOf Good_Hotel
   Hotel_4_Stars and (hasAmenity some Babysitting)
      SubClassOf Good_Hotel
   Bed_and_Breakfast and (hasPrice some hasPrice_high)
      SubClassOf Good_Hotel
```

pgFOIL

```
  Hotel_5_Stars and (hasAmenity some Internet_Access_Point)
      SubClassOf tempConcept_0
   Site_Near_University and (hasAmenity some Disabled_Facilities)
      SubClassOf tempConcept_1
   (hasAmenity some WI-FI) SubClassOf tempConcept_2
   (tempConcept_0 or tempConcept_1 or tempConcept_2)
      SubClassOf Good_Hotel
```

gAdaBoost

```
  (hasAmenity some WI-FI) SubClassOf Good_Hotel_crisp
```

Algorithm	Squared Error	Precision	Recall	$F_1 - score$	Times
Gredy FOIL	0	1	1	1	3659 ms
Best-5 Backtrack FOIL	0	1	1	1	4762 ms
pFOIL	0	1	1	1	28503 ms
gFOIL	0.0025	1	0.9889	0.9943	166871 ms
pgFOIL	0.0217	0.9778	0.916	0.9452	228140 ms
gAdaBoost Weighted	0.0774	1	0.4543	0.6246	825460 ms
gAdaBoost Thresholded	0.0125	1	0.9511	0.9747	825460 ms

Table 8.3: Moral 5-fold cross validation average results.

```
Accomodation and Site_Near_Civic
   SubClassOf Good_Hotel_crisp
Site_Near_Civic and (hasAmenity some Amenity)
   SubClassOf Good_Hotel_crisp
Hostel and (hasAmenity some Cradle)
   SubClassOf Good_Hotel_crisp
(hasAmenity some WI-FI) SubClassOf T_1
Accomodation and Site_Near_Civic SubClassOf T_2
Site_Near_Civic and (hasAmenity some Amenity)
   SubClassOf T_3
Hostel and (hasAmenity some Cradle)
   SubClassOf T_4
0.33*T_1 + 0.24*T_2 + 0.28*T_3 + 0.15*T_4
   SubClassOf weightedSum
weightedSum SubClassOf Good_Hotel
```

8.2.3 The Moral Ontology

This ontology has been tested through k-fold cross validation. The target concept has been set as ToLearn and concept guilty, defined equivalent to ToLearn, has been excluded from the learning experiments. Max depth used was 1 and max length 2. As for hotel ontology, a 5-fold cross validation has been run. Table 8.3 depicts the average performance of algorithm through the 5 runs.

As can be seen from the results, target concept is well explained by samples, in fact results are quite high for almost all of the algorithms. It is clear that FOIL and pFOIL perform better than the others. Algorithms that exploit hybrid learning do not perform perfectly on this ontology. This is mainly due to the recall, in fact recall more than precision causes performances to decrease.

As before weighted gAdaBoost seems to perform badly. Anyway, for the

same reasons explained earlier, this is what was expected.

Following we report the axioms learnt during the first run.

Greedy FOIL

```
blameworthy SubClassOf ToLearn
 vicarious_blame SubClassOf ToLearn
```

Best-5 Backtrack FOIL

```
blameworthy SubClassOf ToLearn
 vicarious_blame SubClassOf ToLearn
```

pFOIL

```
intend_mental_state SubClassOf tempConcept_0
 vicarious_blame SubClassOf tempConcept_1
 blameworthy SubClassOf tempConcept_2
 (tempConcept_0 or tempConcept_1 or tempConcept_2)
    SubClassOf Good_Hotel
```

gFOIL

```
monitor and vicarious_blame SubClassOf ToLearn
 external_cause and intend_mental_state SubClassOf ToLearn
 blameworthy and careful SubClassOf ToLearn
 blameworthy and high_foreseeability SubClassOf ToLearn
 goal_achieveable_less_harmful and vicarious_blame
    SubClassOf ToLearn
 blameworthy and monitor SubClassOf ToLearn
 foreseeable and vicarious_blame SubClassOf ToLearn
 blameworthy and neither_mental_state SubClassOf ToLearn
 blameworthy and negligent_c SubClassOf ToLearn
```

pgFOIL

```
blameworthy and foresee_intervention SubClassOf tempConcept_0
 blameworthy and weak_intend tempConcept_1
 blameworthy and control_perpetrator tempConcept_2
 notaccident and vicarious_blame tempConcept_3
 blameworthy and goal_achieveable_less_harmful tempConcept_4
 (tempConcept_0 or tempConcept_1 or tempConcept_2
    or tempConcept_3 or tempConcept_4) SubClassOf ToLearn
```

gAdaBoost Average time elapsed:707628 ms

```
blameworthy SubClassOf ToLearn_crisp
   blameworthy SubClassOf T_1
   1.0*T_1 SubClassOf weightedSum
   weightedSum SubClassOf ToLearn
```

8.2.4 Trains

This ontology has been tested following the goal oriented approach. Target concepts used were WestTrain and EastTrain. Max depth has been set to 1 and max length to 2.

8.2.4.1 EastTrain

This learning process is executed using EastTrain as target concept.

Greedy FOIL learns

```
   3CarTrain and (hasCar some ClosedCar)
      SubClassOf EastTrain accuracy:1.0
   4CarTrain and (hasCar some TriangleLoadCar)
      SubClassOf EastTrain accuracy:1.0
```

Best-5 backtrack FOIL is able to learn the rule:

```
   hasCar some (ClosedCar and ShortCar)
      SubClassOf EastTrain accuracy:1.0
```

pFOIL learns

```
   Train and (hasCar some (ClosedCar and ShortCar))
      SubClassOf tempConcept_0 accuracy:1.0
   3CarTrain and (hasCar some (2LoadCar))
      SubClassOf tempConcept_1 accuracy:1.0
   (tempConcept_0 or tempConcetp_1)
      SubClassOf EastTrain accuracy:1.0
```

gFOIL learns

```
   3CarTrain and (hasCar some (TriangleLoadCar and 3WheelsCar))
      SubClassOf EastTrain accuracy:1.0
   4CarTrain and (hasCar some (ElipseShapeCar and 2WheelsCar))
      SubClassOf EastTrain accuracy:1.0
```

```
3CarTrain and (hasCar some (ClosedCar and 2WheelsCar))
   SubClassOf EastTrain accuracy:1.0
Train and (hasCar some (HexagonLoadCar and OpenCar))
   SubClassOf EastTrain accuracy:1.0
```

pgFOIL learns

```
hasCar some (ClosedCar and ShortCar) SubClassOf T0 accuracy:1.0
3CarTrain and (hasCar some (CircleLoadCar and ClosedCar))
   SubClassOf T1 accuracy:1.0
(T0 or T1) SubClassOf EastTrain
```

Weighted gAdaBoost learns

```
3CarTrain and (hasCar some (ClosedCar and ShortCar))
   SubClassOf T0 accuracy:1.0
4CarTrain and (hasCar some (RectangleShapeCarTriangleLoadCar and
2WheelsCar))
   SubClassOf T1 accuracy:1.0
3CarTrain and (hasCar some (ClosedCar and RectangleShapeCar))
   SubClassOf T2 accuracy:1.0
4CarTrain and (hasCar some (TriangleLoadCar and 1LoadCar))
   SubClassOf T3 accuracy:1.0
0.33*T0 + 0.25*T1 + 0.21*T2 + 0.21*T3
   SubClassOf weightedSum
weightedSum SubClassOf EastTrain
```

Thresholded gAdaBoost learns

```
3CarTrain and (hasCar some (ClosedCar and ShortCar))
   SubClassOf EastTrain_Crisp accuracy:1.0
4CarTrain and (hasCar some (RectangleShapeCarTriangleLoadCar and
2WheelsCar))
   SubClassOf EastTrain_Crisp accuracy:1.0
3CarTrain and (hasCar some (ClosedCar and RectangleShapeCar))
   SubClassOf EastTrain_Crisp accuracy:1.0
4CarTrain and (hasCar some (TriangleLoadCar and 1LoadCar))
   SubClassOf EastTrain_Crisp accuracy:1.0
```

CELOE and *ELTL* algorithms from *DL-LEARNER* learns `hasCar some (ClosedCar and ShortCar) SubClassOf EastTrain accuracy:1.0`.

Among the algorithms the most satisfactory performance is produced by Best-5 backtrack FOIL and pgFOIL.

8.2.4.2 WestTrain

This learning process is executed using WestTrain as target concept.

Greedy FOIL learns (in 1909 ms)

```
2CarTrain SubClassOf WestTrain accuracy:1.0
4CarTrain and (hasCar some JaggedCar)
   SubClassOf WestTrain accuracy:1.0
3CarTrain and (hasCar some JaggedCar)
   SubClassOf WestTrain accuracy:1.0
```

Best-5 backtrack FOIL is able to learn the rule (in 167 ms):

```
2CarTrain SubClassOf WestTrain accuracy:1.0
3CarTrain and (hasCar some 0LoadCar)
   SubClassOf WestTrain accuracy:1.0
hasCar some JaggedCar
   SubClassOf WestTrain accuracy 1.0
```

pFOIL learns (in 953 ms)

```
2CarTrain SubClassOf T0 accuracy:1.0
4CarTrain and (hasCar some JaggedCar)
   SubClassOf T1 accuracy:1.0
3CarTrain and (hasCar some 0LoadCar)
   SubClassOf T2 accuracy:1.0
4CarTrain and (hasCar some (CircleLoadCar and UShapeCar)
   SubClassOf T3 accuracy:1.0
(T0 or T1 or T2 or T3) SubClassOf WestTrain accuracy:1.0
```

gFOIL learns (in 1483 ms)

```
2CarTrain and (hasCar some (LongCar and RectangleShapeCar))
   SubClassOf WestTrain accuracy:1.0
3CarTrain and (hasCar some (ShortCar and 0LoadCar))
   SubClassOf WestTrain accuracy:1.0
Train and (hasCar some (JaggedCar and RectangleShapeCar))
   SubClassOf WestTrain accuracy:0.33
```

pgFOIL learns (in 2281 ms)

```
2CarTrain and (hasCar some (ClosedCar and 3LoadCar)
   SubClassOf T0 accuracy:1.0
```

```
2CarTrain and (hasCar some (ClosedCar and 3LoadCar))
   SubClassOf T1 accuracy:1.0
(T0 or T1) SubClassOf westTrain
```

Weighted gAdaBoost learns

```
2CarTrain and (hasCar some (ShortCar and 2WheelsCar))
   SubClassOf T0 accuracy:1.0
2CarTrain and (hasCar some (OpenCar and 1LoadCar))
   SubClassOf T1 accuracy:1.0
2CarTrain and (hasCar some (OpenCar and ShortCar))
   SubClassOf T2 accuracy:1.0
2CarTrain and (hasCar some (1LoadCar and 2WheelsCar))
   SubClassOf T3 accuracy:1.0
2CarTrain and (hasCar some (LongCar and RectangleShapeCar))
   SubClassOf T4 accuracy:1.0
2CarTrain and (hasCar some (ShortCar and 1LoadCar))
   SubClassOf T5 accuracy:1.0
0.28*T0 + 0.21*T1 + 0.17*T2 + 0.14*T3 + 0.12*T4 + 0.08*T5
   SubClassOf weightedSum
weightedSum SubClassOf WestTrain
```

Thresholded gAdaBoost learns (in 3793 ms)

```
2CarTrain and (hasCar some (ShortCar and 2WheelsCar))
   SubClassOf WestTrain accuracy:1.0
2CarTrain and (hasCar some (OpenCar and 1LoadCar))
   SubClassOf WestTrain accuracy:1.0
2CarTrain and (hasCar some (OpenCar and ShortCar))
   SubClassOf WestTrain accuracy:1.0
2CarTrain and (hasCar some (1LoadCar and 2WheelsCar))
   SubClassOf WestTrain accuracy:1.0
2CarTrain and (hasCar some (LongCar and RectangleShapeCar))
   SubClassOf WestTrain accuracy:1.0
2CarTrain and (hasCar some (ShortCar and 1LoadCar))
   SubClassOf WestTrain accuracy:1.0
```

CELOE and *ELTL* algorithms from *DL-LEARNER* learns

```
hasCar some (ClosedCar and ShortCar)
   SubClassOf EastTrain accuracy:1.0.
```

CELOE algorithm of *DL-LEARNER* learns the axiom

```
hasCar some LongCar accuracy:0.8
```

ELTL algorithm of *DL-LEARNER* learns the axiom

`hasCar only (LongCar or OpenCar) accuracy 1.0`

The last axiom won't never be learnt by fDLL as it works only on $\mathcal{EL}(D)$ and cannot learn a universal restriction class expression (denoted by only).

However all fDLL algorithms learn rules with accuracy 1, it means that they do not wrongly classify any individual of training set. This can be considered a good result.

Part IV

Conclusions

In this work, we have defined, adapted, implemented and compared several machine learning algorithms to induce inclusion axioms, given an *OWL 2* ontology and a target concept whose descriptive characterisation we were searching for. Some basic ideas for these algorithms have been taken from ILP and have been then adapted to the *OWL 2* context. To improve readability, we also allowed automatically generated fuzzy concepts to occur in the induced axioms. In summary, we may conclude that among all the tested algorithms, the pFOIL algorithm seems to to be the best option so far.

Bibliography

[1] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 364–369, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[2] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. LTCS-Report LTCS-05-01, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html.

[3] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope further. In Kendall Clark and Peter F. Patel-Schneider, editors, *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.

[4] Franz Baader and Philipp Hanschke. A schema for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, 1991.

[5] B. Cuenca-Grau, I. Horrocks, B. Motik, B. Parsia, P.F. Patel-Schneider, and U. Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008.

[6] H. Prade D. Dubois. Gradual Elements in a Fuzzy Set. *Soft Computing*, 12(2):165–175, 2008.

[7] Description Logics Web Site. `http://dl.kr.org`.

[8] Didier Dubois and Henri Prade. Possibility theory, probability theory and multiple-valued logics: A clarification. *Annals of Mathematics and Artificial Intelligence*, 32(1-4):35–66, 2001.

[9] Saso Dzeroski, James Cussens, and Suresh Manandhar. An introduction to inductive logic programming and learning language in logic. In *Learning Language in Logic*, volume 1925 of *Lecture Notes in Computer Science*, pages 3–35, 2000.

[10] J. Chamorro-Martinez et al. A discussion on fuzzy cardinality and quantification. Some applications in image processing. In *Fuzzy Sets Systems*, 2013.

[11] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.

[12] Fuzzy OWL 2 Web Ontology Language . *http://www.straccia.info/software/FuzzyOWL/*. ISTI - CNR, 2011.

[13] Christoph Haase and Carsten Lutz. Complexity of subsumption in the $\mathcal{EL}$ family of description logics: Acyclic and cyclic TBoxes. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikos Avouris, editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI08)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 25–29. IOS Press, 2008.

[14] Petr Hájek. *Metamathematics of Fuzzy Logic*. Kluwer, 1998.

[15] Petr Hájek. Making fuzzy description logics more general. *Fuzzy Sets and Systems*, 154(1):1–15, 2005.

[16] Petr Hájek. What does mathematical fuzzy logic offer to description logic? In Elie Sanchez, editor, *Fuzzy Logic and the Semantic Web*, Capturing Intelligence, chapter 5, pages 91–100. Elsevier, 2006.

[17] Petr Hájek. On witnessed models in fuzzy logic. *Mathematical Logic Quarterly*, 53(1):66–77, 2007.

[18] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.

[19] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*, pages 57–67. AAAI Press, 2006.

[20] Ian Horrocks, Peter F. Patel-Schneider, Deborah L. McGuinness, and Christopher A. Welty. OWL: A description logic based ontology language for the semantic web. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition)*, chapter 14. Cambridge University Press, 2007.

[21] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*. Trends in Logic - Studia Logica Library. Kluwer Academic Publishers, 2000.

[22] Niels Landwehr, Kristian Kersting, and Luc De Raedt. nFOIL: Integrating naïve Bayes and FOIL. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 795–800. AAAI Press, 2005.

[23] Jens Lehmann. Hybrid learning of ontology classes. In Petra Perner, editor, *MLDM*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer, 2007.

[24] Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78(1-2):203–250, 2010.

[25] Francesca A. Lisi and Umberto Straccia. A logic-based computational method for the automated induction of fuzzy ontology axioms. *Fundamenta Informaticae*, 124(4):503–519, 2013.

[26] Francesca A. Lisi and Umberto Straccia. A system for learning gci axioms in fuzzy description logics. In *Proceedings of the 26th International Workshop on Description Logics (DL-13)*, volume 1014 of *CEUR Workshop Proceedings*, pages 760–778. CEUR-WS.org, 2013.

[27] Francesca Alessandra Lisi and Umberto Straccia. Dealing with incompleteness and vagueness in inductive logic programming. In *28th Italian Conference on Computational Logic (CILC-13)*, volume 1068, pages 179–193. CEUR Electronic Workshop Proceedings, 2013.

[28] Carsten Lutz. Reasoning with concrete domains. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 90–95. Morgan Kaufmann Publishers Inc., 1999.

[29] Carsten Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logics Volume 4*. King's College Publications, 2003.

[30] Richard Nock and Frank Nielsen. A real generalization of discrete adaboost. *Artif. Intell.*, 171(1):25–41, January 2007.

[31] OWL Web Ontology Language overview. *http://www.w3.org/TR/owl-features/*. W3C, 2004.

[32] OWL 2 Web Ontology Language Document Overview. *http://www.w3.org/TR/2009/REC-owl2-overview-20091027/*. W3C, 2009.

[33] OWL 2 Web Ontology Language Profiles. *http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/*. W3C, 2009.

[34] OWL 2 Web Ontology Language Profiles: OWL 2 EL. *http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/#OWL_2_EL*. W3C, 2009.

[35] J. R. Quinlan. Learning logical definitions from relations. *MACHINE LEARNING*, 5:239–266, 1990.

[36] Umberto Straccia. Description logics with fuzzy concrete domains. In Fahiem Bachus and Tommi Jaakkola, editors, *21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 559–567, Edinburgh, Scotland, 2005. AUAI Press.

[37] Umberto Straccia. *Foundations of Fuzzy Logic and Semantic Web Languages*. CRC Studies in Informatics Series. Chapman & Hall, 2013.

[38] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

Appendix A

Some Implementation Details

To work with aforementioned learning algorithms a learning system called *fDLL* has been developed. In this chapter fDLL will be roughly described. First of all, we will talk about the description logic API created, then we will talk about how OWL ontologies and DLs are used throughout this work. To manage ontologies we used the OWL API ([18]) that can be used to deal with reasoners. In this way it is possible to perform instance checking, instance retrieval, consistency checking, etc.. However the OWL API cannot deal with fuzziness. fDLL implements an extended API that deals with fuzzy degree of truth. Moreover fDLL's API gives the chance to define new fuzzy datatypes and use them in data restrictions.

A.1 Description Logic API

To manage description logics background fDLL extends the OWL API with fuzzy degree of truth. Each ClassExpression C will have a method `degree(a)` that outputs a double number in $[0, 1]$ representing the $bed(\mathcal{K}, a : C)$. Available ClassExpression are the ones used for refinement and the disjunction, used to produce appropriate results for pFOIL.

So we have that a ClassExpression C can be:

1. A, being A an atomic concept;
2. $\exists R.D$, being R an atomic object property and D a ClassExpression;
3. $\exists S.d$, being S an atomic data property and d a fuzzy datatype;
4. $C_1 \sqcap ... \sqcap C_n$ being $C_1, ..., C_n$ ClassExpressions

It is important to note that data properties are only used together with fuzzy datatypes defined within fDLL. So a data property will not be used unless a fuzzy datatype has been defined on it. Datatypes already defined in the ontologies will be ignored.

It can be easily noticed that fuzziness can only be introduced by datatypes. However they can spread fuzziness both to object restrictions and to conjunction. In fact we can have an object restriction $\exists R.(\exists S.d)$, where R is an object property, S a data property and d a fuzzy datatype. In this case the restriction becomes fuzzy. Similarly if a conjunct of a conjunction is fuzzy, *i.e.*, a data restriction, the whole conjunction becomes fuzzy.

A.1.1 Fuzzy Datatypes

Fuzzy datatypes are defined by a fuzzy function (see Table 2.1). In this work fuzzy functions are obtained through two different methods.

Triangular Same Width Discretizer: it simply splits the interval $[min_S, max_S]$ of a data property S into a predefined number of equidistant points. min is the minimum values related to an individual from the ontology while max is the max value related to an individual, *i.e.*, $min_S = min_{a \in \mathcal{I}}\{r \in [0,1] \mid (a,r) : S\}$ and $max_S = max_{a \in \mathcal{I}}\{r \in [0,1] \mid (a,r) : S\}$, where S is a data property and $\mathcal{I}$ is the set of all individuals of ontology. So if we want to have n datatypes we would split the interval $[min_S, max_S]$ into $n+1$ equidistant points. We will then define $n-2$ triangular functions, a left shoulder and a right shoulder. We will have that I_0 will be a left shoulder $ls(min, p0)$, I_i will be a triangular function $tri(p_{i-1}, p_i, p_{i+1})$ with $i = 1, ..., n-2$, and finally I_{n-1} will be a right shoulder $rs(p_{n-2}, p_{n-1})$.

Trapezoidal Same Width Discretizer: it simply splits the interval $[min_S, max_S]$ of a data property S into a predefined number of equidistant points. min is the minimum values related to an individual from the ontology while max is the max value related to an individual, *i.e.*, $min_S = min_{a \in \mathcal{I}}\{r \in [0,1] \mid (a,r) : S\}$ and $max_S = max_{a \in \mathcal{I}}\{r \in [0,1] \mid (a,r) : S\}$ where S is a data property and $\mathcal{I}$ is the set of all individuals of the ontology. So if we want to have n datatypes we would split the interval $[min_S, max_S]$ into $n+1$ equidistant points. We will then define $n-2$ trapezoidal functions, a left shoulder and a right shoulder. We will have that I_0 will be a left shoulder $ls(min, p0)$, I_i will be a triangular function $tri(p_{i-1}, p_i, p_{i+1}, p_{i+2})$ with $i = 1, ..., n-2$ and $p_n = max$, finally I_{n-1} will be a right shoulder $rs(p_{n-2}, p_{n-1})$.

Naturally every discretization can be adopted to create fuzzy functions for a data property. However a particular care should be adopted if trying to use discretization methods guided by a target concept. In this work the latter kind of discretization are not considered also because they can bias the learning.

Beside discretization methods, fDLL provides a simple GUI to manage fuzzy datatypes. It is possible to define brand new fuzzy functions cloning an existing fuzzy datatype and modifying it.

A.1.2 Reasoners and reasoning tasks

As explained earlier, OWL API provide an interface that can work with reasoners. fDLL can use three reasoners, PELLET[1], HERMIT[2] and JFACT[3], however, all tests have been carried on using PELLET.

Thanks to reasoners, it is possible to perform reasoning tasks though some care has to adopted to get reasonable response time.

Specifically, fDLL's implementation of logic api buffers all individuals when the first reasoning operation is requested, whatever the operation is. In fact *OWL 2* reasoners are very fast when performing instance retrieval. This approach is by far more fast than performing a certain number of instance checking. Few instance checking, probably less than 10, will cost more than a full instance retrieval.

Once instance retrieval is performed on a concept, instances are buffered together with their degree of membership. For crisp concepts the degree will always be 1. For fuzzy concepts, *i.e.*, concepts that involves existential data restrictions on fuzzy datatypes, only individuals with membership degree bigger than 0 are buffered.

A.2 Refinement Operator Implementation

Refinement operator usage is the most time consuming task, so it is necessary to pay particular attention on it. Refinement operators are an implementation of an interface `AbstractRefine` that provides a simple method with signature:

```
public ArrayList<ClassExpression> refine(ClassExpression ce)
```

It is a duty of implementation to deal with performance.

[1] `http://clarkparsia.com/pellet/`
[2] `hermit-reasoner.com/`
[3] `http://jfact.sourceforge.net/`

fDLL implementation of `AbstractRefine` organizes refinement of class expression in a structure similar to direct acyclic graphs with links managed by graph nodes. The graph obtained is acyclic because it is not possible to obtain a concept refining one of its refinement.

To avoid duplication, nodes creation has been centralized. Once a node is created it is inserted into an `HashMap` using as key the result of method:

```
public String toStringID()
```

This method is declared on the logic API interface for `ClassExpression` so every class expression have to implement it. It is necessary that the string returned by this method uniquely identifies a class expression.

A further request made on this method is that if two class expressions should be considered equal, then they should return the same id, *e.g.*, if two conjunctions have the same conjuncts but in different orders, they can be considered equal, if it is the case they should have the same id. In our logic API implementation conjunctions are ordered, *i.e.*, before a conjunction is created the conjuncts used for its creation are ordered with respect to their id.

The main class of refinement operator, `RefineGraph`, provides a method for generic node creation from a generic class expression, a different node is available for each type of class expression. New nodes are created through an implementation of a visitor interface provided by the logic API.

When a method in `RefineGraph` is asked to create a new node, it first checks if a node for the class expression has been previously created. If not, it invokes the creation method of the aforementioned visitor.

The refinements returned are only those that have a positive witness. However during the refinement process all refinements are evaluated, also those without positive witness and all the refinements are buffered unless they are inconsistent. There's no use in saving inconsistent refinements as they will never be useful for learning. Instead concepts without positive witness, or without witness from ontology, could be useful as role filler.

Summing up, when we have a `RefineGraph` object, that implements `AbstractRefine`, and we ask it to refine a class expression, it performs the following steps:

1. It checks whether or not there is an available node that refers to that class expression, *i.e.*, if a node has been buffered and associated to the same id of the class expression;

2. If such a node is already buffered, it is asked to provide its refinements. If refinements are available the node simply returns them, otherwise the node will compute its refinements, *i.e.*, those relative to the class

expression represented, it will buffer them and returns the list with such refinements;

3. The `RefineGraph` object will return the list of refinements provided by the class expression node.

A.3 Algorithm Implementation

All algorithms previously described have been implemented. As they all exploit a refinement operator they all use the refinement operator defined by `RefineGraph` described in Section A.2. Moreover all algorithms ask for a `ChunkBuffer` in such a way they can send learnt axioms through this synchronized buffer. This object can be null, in that case axioms are not sent.

A.3.1 DL-FOIL

DL-FOIL has been implemented following Algorithm 4.5. Both greedy and best-k backtrack versions have been implemented, however the difference between the two have been handled through handlers.

The main class for DL-FOIL is `FOIL` contained in package `fDLL.foil`. It implements the algorithm in both its part, `learnSetOfAxioms` and `learnOneAxiom`. The `FOIL` object includes informations about the logic as a factory to produce class expressions and to manage ontology interaction, the target concept, the lists of individuals, atomic concepts, object properties, data properties and datatypes available for learning. It also needs parameters like maximum depth, maximum length, θ, positive coverage, negative coverage and a flag used to indicate whether or not open world is assumed. Beside these informations, the class also asks to have a `HandlerSelector`, an `AbstractConfidence`, an `AbstractGain` and an `AbstractRefine`. The latter object refers to a refine object, as explained earlier. The others will be explained in the following.

`AbstractConfidence.`

`AbstractConfidence` is an interface that defines a confidence function. It provides a method

```
public double computeConfidence(ClassExpression ce,
  ArrayList<Individual> positives,
  ArrayList<Individual> negatives)
```

that computes the confidence of class expression ce w.r.t. individuals contained in variable `positives` considered as positive samples and individuals contained in variable `negatives` as negative samples.

The implementation used by fDLL simply implements the confidence function defined in Definition 9.

`AbstractGain`.

`AbstractGain` is an interface that defines an abstraction of a gain method and provides method

```
public double computeGain(ClassExpression oldCE,
  ClassExpression newCE,
  ArrayList<Individual> positives,
  ArrayList<Individual> negatives)
```

that is intended to compute the gain of newCE w.r.t. oldCE considering as positive sample the individuals contained in `positives` variable and as negative samples the individuals contained in `negatives` variable.

SamplesHandler. Finally `FOIL` objects needs a `HandlerSelector`. Handler selectors are objects that build a SamplesHandler. So `HandlerSelector` and classes implementing it are factories. `SamplesHandler` is an interface that implements strategies to manage samples. `SamplesHandler` object may or not have a backtrack method.

```
public boolean hasLocalBacktrack();
```

returns true if the handler exploits a backtrack policy. Samples handler also manages the removal of positive samples during `learnSetOfAxiom`, the setting of negative samples during `learnOneAxiom`, the evaluation of class expressions and the checks on positive and negative coverages. `FOIL` objects completely abstract from sample handling and delegate this task to samples handler. `FOIL` objects simply asks for backtrack after having verified that a handler exploits such a policy and asks the handler to perform all operations concerning samples manipulation.

A.3.2 pFOIL

The main class for pFOIL is `PFoil`, contained into package fDLL.pfoil. This class is more simple than `FOIL` class as it does not need samples handlers and backtrack is used by default. Like `FOIL`, `PFoil` objects are built using logic informations and parameters as above. The difference stands in evaluators. Differently from `FOIL`, `PFoil` does not need confidence and gain measures

as it uses probability measures to evaluate class expressions. So, an object typed `AbstractProbability` is requested. `AbstractProbability` provides a method to both access individuals and to compute two types of scores. In fact pFOIL is implemented exploiting two different scores, one used to compare class expressions in `learnOneAxiom` and the other one to evaluate the ensemble performance in `learnSetOfAxioms`. The latter is mainly necessary to evaluate the stop condition. In fact we assume to stop pFOIL algorithm when we do not achieve an increment in score above a certain threshold or when it is no more possible to obtain a "legal" class expression. In the first case a measure to evaluate ensemble performance is needed.

In our implementation both performances are evaluated through the $F_\beta - score$, as explained in Section 5.2. The only difference between the two stands in the fact that they use different values of β.

A.3.3 gFOIL and pgFOIL

Hybrid Learning algorithms are implemented through the `HLAlgorithm` class. This class needs like the others logic informations and some parameters. It also need some parameters used during genetic learning. Parameters needed are: a number indicating the number of genetic iterations to perform (`nIter`), an integer indicating the size of the population (`popSize`) a selection ratio (`ps`) indicating the fraction of the actual population that will be nominated to reproduction, a cross over probability (`pc`) indicating the probability that during reproduction a crossover is used, a refine probability (`pr`) indicating the probability of performing refinement during reproduction, and a probability of mutation (`pm`) determining the probability of performing a mutation on a population member after reproduction.

It can be noticed that `pc` and `pr` are strictly related. In fact they refer to a choice between two different types of operation. As they are complementary, their sum should be at most 1. If it is the case, then, with probability `pc`, a crossover is performed, with probability `pr` a refine is performed and with probability 1-(`pc`+`pr`) a class expression is simply forwarded to the successive population.

Our hybrid learning algorithm needs three operators: a refinement operator, a crossover operator and a mutation operator. The first two are not requested from class constructor while the latter is. The first two are automatically built when needed. However a setter method give the possibility to use particular refinement or crossover operator.

The refinement used is the same as `FOIL` and `PFoil`. Crossover operators instead are objects of type `CrossOver` that is an interface providing a method

```
public ArrayList<ClassExpression> cross(ClassExpression c1,
  ClassExpression c2)
```

The implementation used by fDLL exploits several wrappers extending interface `CrossableClassExpression` which provides methods:

```
public CrossableClassExpression getRightSplit()
public CrossableClassExpression getLeftSplit()
public CrossableClassExpression cross(
  CrossableClassExpression cce)
```

the first two methods split the class expression referred by the wrapper and the latter cross cce with class expression referred by the wrapper. The implementation exploited by fDLL follows the definition presented in Section 6.1.2. Each type of class expression available has a proper wrapper. Wrappers are created through an implementation of a logic visitor as for refinement.

Mutator operators instead implement the interface `Mutator` providing method

```
public ClassExpression mutate(ClassExpression ce)
```

that mutates ce. fDLL implementation of `Mutator` follows the definition of mutation provided in Section 6.1.3.

Last parameter is how class expressions are evaluated. As for refine and crossover, a default evaluator is available and a set method is used to set another evaluator.

gFOIL and pgFOIL. These two algorithms are implemented respectively in class `GFoil` and `GPFoil`. These classes simply extend respectively `FOIL` and `PFoil` overriding `learnOneAxiom` method. This time `learnOneAxiom` simply creates a `HLAlgorithm` instance that evaluates class expressions using a confidence measure on available individuals. Once `learnOneAxiom` is run, `learnSetOfAxioms` removes covered positive samples and at the next iteration `learnOneAxiom` will create an instance of `HLAlgorithm` that considers only positive samples not yet removed.

`GPFoil` extends `PFoil` overriding `leanrOneAxiom`. Here the `HLAlgorithm` object exploits an evaluator that considers ensemble performance.

A.3.4 gAdaBoost

The gAdaBoost algorithm is implemented through `GeneticBoost` class from package `fDLL.boost`. As explained earlier, gAdaBoost iterates a certain number of times (this number is a parameter) and at each iteration learns

an axiom exploiting hybrid learning and so `HLAlgorithm`. At each iteration the only modification occurring in `HLAlgorithm` are about individual weighting. To implement this kind of scenario, an evaluation function that considers individual weights has been introduced. The interface used to perform evaluation is called `WeightedIndividualEvaluator` and extends `IndividualEvaluator`, the interface used for`HLAlgorithm` evaluation functions. `IndividuaEvaluator` provides the method

```
public double evaluateIndividual(ClassExpression ce,
  ArrayList<Individual> positives,
  ArrayList<Individual> negatives)
```

used to evaluate class expression ce (that is an individual of the population used in `HLAlgorithm`) w.r.t. individuals of variables `positives` and `negatives` respectively considered as positive and negative samples.

`WeightedIndividualEvaluator` extends the interface providing the method

```
public void updateWeights(ArrayList<WeightedIndividual> inds)
```

used to update weights w.r.t. weights defined in `inds`.

So, `WeightedIndividualEvaluator` has a method to evaluate a class expression and a method to update weights of the individuals. In our implementation, `updateWeights` is used to buffer individuals with weights.

`GeneticBoost` at each iteration let a `HLAlgorithm` object perform hybrid learning, once a population is returned, `HLAlgorithm` is asked to choose the best class expression from such a population. The chosen class expression is learnt and weights are updated considering the learnt class expression as the classifier. Next `HLAlgorithm` is run using as evaluator the `WeightedIndividualEvaluator` with updated weights.

A.3.5 K-Fold Cross Validation

To perform K-Fold Cross Validation fDLL has class `KFoldCrossValidation` from package `fDLL.kFoldCrossValidation`. This class is built with logic informations and a parameter k representing the number of folds. If the number of individuals is smaller or equal to k than a *leave-one-out* cross validation is performed.

The `KFoldCrossValidation` object has a method for each learning algorithm, essentially a split of samples set (containing both positive and negative samples) into k subsets is performed. Once a cross validation of an algorithm is called, the algorithm is performed k times. Each time one of the k split is used as test set while the others are used as training set.

At the end, an `ArrayList` containing k objects of type `KFoldResult` is returned. A `KFoldResult` object refers to a run of `KFoldCrossValidation`. It contains the training set and the test set used for the run, an `ArrayList` containing the learnt axioms and provides the squared error, the precision, the recall and the $F_1 - score$, computed from the test set. The squared error is computed as follows:

$$e(\mathcal{I}) = \sum_{a \in \mathcal{I}} (\mathcal{H}(a) - T(a))^2 \ ,$$

where $\mathcal{I}$ is the test set, $\mathcal{H}$ is the set of axioms learnt and $\mathcal{H}(a)$ is the predicted value for a, *i.e.*, with which degree a belongs to T following the prediction made by $\mathcal{H}$. $T(a)$ instead is the actual degree with which a belongs to T.