Semantic coordination of hierarchical classifications with attributes

S. Sceffer L. Serafini S. Zanobini

December 6, 2004

Contents

Chapter 1

Introduction

In this paper, our purpose is to improve the CtxMatch algorithm (i) in the type of the schemas considered, (ii) in the way we make the meaning of the elements explicit and (iii) in the way we retrieve information from knowledge bases.

Firstly, we extend the methodology to hierarchical classifications with *attributes*. Intuitively, each element of a schema can be associated with a set of attributes which specify the features that the objects classified under that element must own. For example, it could be required for all the images under the nodes `Images` to specify their format, if they are multi–colored or black–and–white, the corresponding subject and the place where they have been taken. Note that attributes delineate the meaning of the nodes. For example, the node `Mountain` on the left would represent 'images of a given format in multi– or black–and–white colors whose subject is Tuscan mountains and which have been taken in a certain place'. Moreover, the selection of a precise value for each attribute further restricts the node meaning. For example, it may become 'large multi–colored images of Tuscan mountains taken in Florence', where 'large' represents the image format.

The second improvement we present is the use of a different formal language to represent the meaning of the nodes. CtxMatch was essentially based on propositional logic. Therefore, the actual intepretation of the node `Mountain` on the left was 'images and Tuscany and mountain'. The different concepts were simply connected by the operator of conjunction. In this thesis, we discuss the possibility of fully exploiting the expressive power of Description Logic to formalize the meaning of the nodes. In fact, we employ *DL roles* and the corresponding *constructors* in order to semantically connect the schema concepts which are involved in the meaning of a certain element. Intuitively, we expect to interpret the node `Mountain` on the left as 'images about mountains located in Tuscany', where 'about' and 'located' represent the roles between the concepts 'images' and 'mountains' and between 'mountains' and 'Tuscany', respectively.

The need of relating elements has motivated the third innovation presented in this thesis, i.e. a methodology for drawing possible *sematic relations* between concepts from existing knowledge bases. For example, given the two concepts 'photo' and 'mountain', we aim at deriving that a 'photo' may *show* a 'mountain' or that the 'photo' could have *been taken on* a 'mountain' top. In particular, we have developed a method to extract complex relations from ontologies expressed in the Description Logic (or equivalent) language.

The paper is organized as follows: Chapter 2 defines the type of schemas we consider. Chapter 3 defines the matching problem, introduces the possibility of solving it by means of semantic methods and presents a notion of soundness which aims at representing a guideline for the evaluation of schema matching techniques. Chapter 4 presents our own methodology in the scope of semantic methods and Chapter 5 illustrates our approach to the approximation of semantic relations between concepts. Chapter 6 introduces the steps of the algorithm we apply to coordinate two given structures. Chapter 7 presents the system architecture and the implementation details of our prototypal application whose tests on a simple case are discussed in Chapter 8. Chapter 9 presents the related works in the field of semantic explicitation. Finally, Chapter 11 draws the conclusions of the whole work.

Chapter 2

Hierarchical classifications with attributes

2.1 Overview

It would be a great advantage to be able to coordinate schemas in general, i.e. with no regards to any particular model instance. Anyway, semantic methods need to consider the purpose of the schemas they analyze since such information is used to compute the meaning of the schema elements. Indeed, although the basic steps of a semantic method (i.e. semantic explicitation and semantic comparison) are general, the way of performing them depends on the type of schema and on the intended use. In this thesis, we focus on *hierarchical classifications with attributes*. Their practical purpose is to *classify* objects (e.g. documents) according to their topic.

Hierarchical classifications are one of the most widely spread models of data. For example, file systems, web open directories (e.g. Google^{TM} and Yahoo!^{TM}) and most of the electronic catalogs for B2B and B2C commerce are hierararchical classifications. Moreover, some of the most important standards for the classification of products and services as Unspsc [14], ECl@ss [3], Kompass [4] and Naics [7] are based on hierarchical schemas. Attributes are not largely considered in such real cases yet. Indeed, as far as we know, ECl@ss is the only categorization system which uses a standardized set of attributes (SSA, i.e. Standard Set Of Attributes). For example, it identifies the following basic set of attributes for the category TELECOMMUNICATION DEVICES (code 19-06-92-90): *Article number*, *EAN code*, *Manufacturer's name*, *Product name* and *Product type description*.

Furthermore, hierarchical classifications are tree schemas essentially, thus representing a very general model of data since many other schemas can be easily converted to such a model. Relational schemas are examples of tree schemas where nodes act as tables, columns, user–defined types, etc. XML schemas with no shared elements can be also turned to tree schemas: tree nodes represent

XML elements and attributes[1]. Anyway, tree schemas do not represent the most general model employed in real cases. Indeed, they cannot share sub–structures or have referential constraints and there is no way of reducing such properties to the available features of the tree schemas[2].

Hierarchical classifications are schemas used for grouping objects into categories. They are tree schemas, where nodes represent categories describing, in some way, the objects they contain. Different kinds of objects can be categorized: for example, tuples if schemas describe relational databases, files if schemas are file systems, images if we are dealing with photography repositories or documents in the case of digital archives. Nodes can be associated with sets of attributes. Attributes indicate the features which are required for the objects in order to be classified under the node. For example, we could require that each image is associated with its size and with its colors or that each file specifies its dimension and its extension, etc.

In hierarchical classifications, nodes are labeled with natural language statements which are composed of words and, possibly, separators between them. They are taken from a wide variety of linguistic expressions and can be single common words (e.g. 'seaside' and 'mountain'), noun phrases (e.g. 'last minute offers') and expressions containing conjunctions (e.g. 'insurance and lost baggage'). More complex formulae can be composed of prepositional phrases, verb phrases and punctuations. Unfortunately, natural language strings are not the only expressions which are used to tag schemas. Indeed, users often prefer acronyms, abbreviations and compact notations to identify (complex) concepts. Futhermore, the range of possible values of compact notations is unlimited in practice. By looking through schemas of product classifications and ontologies, we can easily meet expressions as 'Public-Company', 'CMU_Publication_Entry' and 'BomberPlane'[3], where words are separated by special characters and capital letters.

Each node can be associated to a concept which is derived by analyzing the labels during the browsing of the schema from the root node to the node under exam. Intuitively, an object is classified under a certain node, if it is *about* the concept which is represented by that node[4]. An example will help to clarify the situation.

Figure 2.1 shows a hierarchy which is used by a travel agency to classify its vacation offers and the services it makes available for customers. The agency has identified six different categories which are represented by the six nodes: `Vacations`, `Reports`, `Last minute offers`, `Insurance and lost baggage`, `Seaside` and `Mountain`. Such nodes are arranged into a schema, and the category they represent is given by the analysis of the path from the root to the node itself. For example, the root node `Vacations` contains documents *about* 'holidays' and the node `Reports` contains documents concerning 'reports of past vacations'. The

[1] See Madhavan, Bernstein et al. [34] for an example of conversion.

[2] A possible generalization of tree schemas in order to consider these properties is presented in Madhavan, Bernstein et al. [34].

[3] Taken from the ontology library of the DAML site [2].

[4] This intuition will be formalized in Section 3.3.

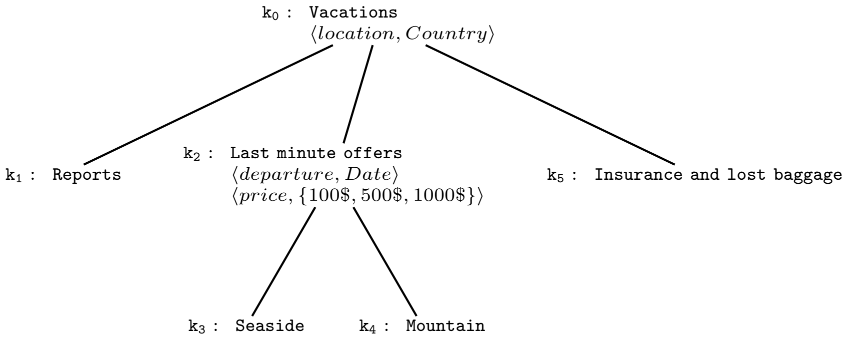

k$_0$: Vacations
$\langle location, Country \rangle$
k$_1$: Reports
k$_2$: Last minute offers
$\langle departure, Date \rangle$
$\langle price, \{100\$, 500\$, 1000\$\} \rangle$
k$_5$: Insurance and lost baggage
k$_3$: Seaside
k$_4$: Mountain

Figure 2.1: A Hierarchical Classification with Attributes

same for the other nodes: `Last minute offers` for 'last minute offers of vacations', `Insurance and lost baggage` for proposals of 'insurance policies for vacations', `Seaside` and `Mountain` for 'last minute offers for seaside destinations' and 'last minute offers for mountain destinations', respectively. We also expect that all the documents in the node `Vacations` specify one value for the attribute *location* in the respective range, i.e. *Country*. Its value is supposed to express the country where the vacation takes place. In other words, the node `Vacations` contains documents about 'offers of holidays to be spent in a given country'. The same consideration holds for the nodes `Reports` and `Insurance and lost baggage`. Similarly, documents in the node `Last minute offers` should specify a value for the attribute *location* in the range *Country* and a value for the attribute *departure* in the range *Date*. The attribute *departure* is supposed to indicate the departure date of the holiday. Moreover, such documents are associated to one of the specified values (namely, 100$, 500$ and 1000$) in order to indicate the offer price, represented by the attribute *price*.

2.2 Formal definition

In this section, we introduce the notions of *hierarchical classification*, *attribute* and *hierarchical classification with attributes*. The corresponding definitions will refer to the whole set of non–empty strings composed of general and commonly used expressions with the notation $\mathcal{NL}$.

Definition 1 (Hierarchical classification) *A* hierarchical classification *(CH) is a triple* $H = \langle K, E, \mathrm{lab} \rangle$*, where* $\langle K, E \rangle$ *is a finite rooted tree and* lab : $K \to \mathcal{NL}$ *is a function that tags each node with a string in* $\mathcal{NL}$*.*

A hierarchical classification is simply a tree where each node is labeled with a natural language string (or in general with an identifying expression).

Definition 2 (Attribute) *An* attribute *is a pair $a = \langle id, F \rangle$, where id is a string in $\mathcal{NL}$ and F is either the empty set, $\emptyset$, or a string in $\mathcal{NL}$, F_r, or a set of strings in $\mathcal{NL}$, F_f.*

An attribute $a = \langle id, F \rangle$ is a pair of elements. The former represents the *attribute name*, while the latter is the *attribute filler.* The filler can express either the generic *filler range*, F_r, i.e. the type of the filler, or the definite *filler range*, F_f, i.e. the possible *values* of the filler. An attribute has a name necessarily, but it can lack an explicit filler, i.e. F is the empty string.

Example 1 *The following pairs represent possible attributes:*

- $a_1 = \langle$*location, Country*$\rangle$;
- $a_2 = \langle$*departure, Date*$\rangle$;
- $a_3 = \langle$*departure, $\emptyset$*$\rangle$;
- $a_4 = \langle$*price, \{100\$, 500\$, 1000\$\}*$\rangle$.

The attribute names are 'location', 'departure' and 'price', respectively. The first attribute indicates 'Country' as filler range. Intuitively, we are asserting that the location must be specified in terms of a country. Similarly, the second case indicates that the filler of the attribute departure[5] *must be a date. In the third case, the filler is empty meaning that no specific type of filler is required for this attribute. In the last case, the filler is expressed by its possible values. Therefore, the price is forced to assume one of the specified values, namely,* 100\$*,* 500\$ *or* 1000\$*.*

We now define hierarchical classifcation with attributes as CH where nodes can be associated with sets of attributes.

Definition 3 (Hierarchical classification with attributes) *Let A be a set of attributes. A* hierarchical classification with attributes *(ACH) is a 4-tuple $H = \langle K, E, \mathrm{att}, \mathrm{lab} \rangle$, where $\langle K, E \rangle$ is a finite rooted tree,* $\mathrm{att} : K \rightarrow 2^A$ *is a function from the set of nodes to a (possibly empty) set of attributes, and* $\mathrm{lab} : K \rightarrow \mathcal{NL}$ *is a function that tags each node with a string in $\mathcal{NL}$.*

A hierarchical classification with attributes is a rooted tree, where each node k is associated with a label, *lab*(k), and with a set of attributes, *att*(k). *att*(k), for a given node k, may be the empty set, i.e. k has no attributes. If *att*(k) is the empty set for each k in K, the ACH becomes a hierarchical classification (CH). Note that we allow attributes to be attached to the nodes of the structure at each level.

[5] In the following, we will refer to an attribute $\langle id, F \rangle$ with the corresponding name, i.e. *id*, where no ambiguity exists about the attribute we are considering.

Example 2 *Figure 2.1 represents an ACH with 6 nodes, namely, k_0, k_1, k_2, k_3, k_4, k_5. The nodes are connected by means of edges resulting in a tree. k_0 is the* root *node and k_1, k_3, k_4, k_5 are called* leaves. *k_3 is said to be a* child *of k_2 which conversely represents its* parent. *Each node in a hierarchy has* one *parent (except for the root node which has no parents). Both k_0 and k_2 are said to be* ancestors *of the node k_3. All of them belong to the same* E–path. *Similarly, this can be said for all the other nodes and for all the other paths.*

The lab *function applied to the nodes of the ACH returns the following strings:*

$$\begin{aligned}
\mathrm{lab}(k_0) &= \textit{Vacations}\\
\mathrm{lab}(k_1) &= \textit{Reports}\\
\mathrm{lab}(k_2) &= \textit{Last minute offers}\\
\mathrm{lab}(k_3) &= \textit{Seaside}\\
\mathrm{lab}(k_4) &= \textit{Mountain}\\
\mathrm{lab}(k_5) &= \textit{Insurance and lost baggage}
\end{aligned}$$

Consider the set of attributes A defined in Example 1. The att *function applied to the ACH nodes returns the following sets of attributes:*

$$\begin{aligned}
&\mathrm{att}(k_0) = \{\langle \textit{location, Country}\rangle\}\\
&\mathrm{att}(k_1) = \emptyset\\
&\mathrm{att}(k_2) = \{\langle \textit{departure, Date}\rangle,\ \langle \textit{price, } \{\textit{100\$, 500\$, 1000\$}\}\rangle\}\\
&\mathrm{att}(k_3) = \emptyset\\
&\mathrm{att}(k_4) = \emptyset\\
&\mathrm{att}(k_5) = \emptyset
\end{aligned}$$

k_0 and k_2 are the only nodes which are provided with a non–empty set of attributes.

2.3 Classification

In this section, we provide a formal definition of the notion of *classification* of a set of objects in a certain ACH. We allow an object to be classified under one node of the structure only. All the nodes of the schema can contain objects and attributes are inherited by all the descendants of the node to which they are attached.

Definition 4 (ACH Classification) *A* classification *of a set of objects D in an ACH $H = \langle K, E, \mathrm{att}, \mathrm{lab}\rangle$ is a pair $\langle \tau, \alpha\rangle$, where $\tau : D \to K$ is a partial function from the set of objects to the set of nodes and $\alpha : D \times A \to \mathcal{NL}$ is a partial function from pairs object–attribute, $d \in D$ and $a \in A$, to strings in $\mathcal{NL}$. α is defined as follows: if $\tau(d) = k$ and $a = \langle id, F\rangle$ is an attribute of k, $a \in \mathrm{att}(k)$, or of an* ancestor *k' of k, $a \in \mathrm{att}(k')$, then:*

$$\alpha(d, a) = \begin{cases} v \in F_f & \textit{if } F = F_f;\\ s \in \mathcal{NL} & \textit{otherwise.}\end{cases}$$

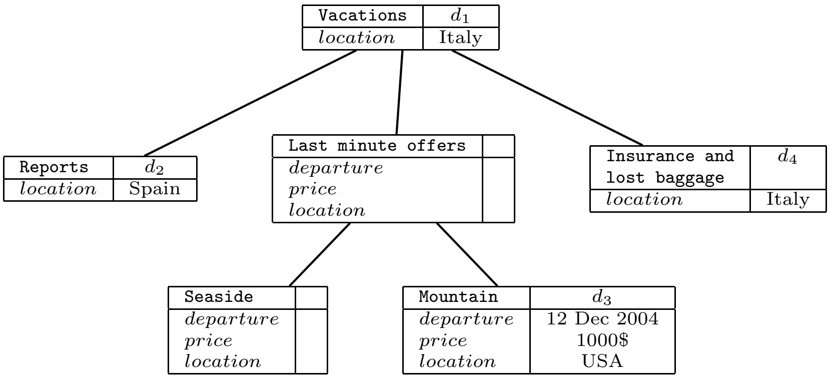


Figure 2.2: Classification of a set of objects in an ACH

A classification classifies a set of objects under the nodes of an ACH. Note that it is not required that each object in D is placed in a node (i.e. the function τ is partial). This corresponds to the intuition that a certain ACH is not right for classifying any kind of object. Each object in an ACH is also required to specify a value for each filler of a set of attributes established by the ACH. In particular, for each object in the ACH, a value must be specified for each attribute attached either to the node k where it is classified or to an ancestor of k. If the attribute does *not* specify the filler (i.e. $F = \emptyset$) or if it specifies its range type only (i.e. $F = F_r$), the attribute value can be a generic non–empty string, whereas it must be a value in F_f, if the possible values of the filler are explicit.

Example 3 *Suppose we have a set of documents $D = \{d_1, d_2, d_3, d_4\}$ to be classified in the ACH of Figure 2.1, where:*

- *d_1 is a generic offer of vacation in Italy by the travel agency;*
- *d_2 is a report of a past vacation in Spain;*
- *d_3 is a last minute offer for a vacation on the USA mountains; the departure date is set on the 12th of December and its price is 1000\$;*
- *d_4 is a possible insurance policy for holidays in Italy.*

Figure 2.2 represents a possible classification of D. Intuitively, each document will be classified under the node best representing its topic. In this case, d_1 is placed into the node `Vacations` *because it is an holiday offer, d_2 into the node* `Reports` *because it is a report of a past vacation, etc. Note that if d_1 were a last minute offer of an holiday, it would be classified into the node* `Last minute offers` *which best represents that category. Formally, τ is defined as follows:*

$$\begin{array}{l} \tau(d_1) = k_0 \\ \tau(d_2) = k_1 \\ \tau(d_3) = k_4 \\ \tau(d_4) = k_5 \end{array}$$

Each document is associated to a set of attribute values according to its position in the hierarchy. For example, the filler value of d_1 for the attribute location *is 'Italy' (which we assume to be a country, since 'Country' represents the type of the filler*[6]*). In fact, d_1 is an holiday offer in Italy. The document d_3 specifies the value for the attribute* location *because* `Mountain` *(i.e. the node where d_3 is classified) is a descendant of* `Vacations` *and for the attributes* departure *and* price *because it is a descendant of* `Last minute offers`*. Similar considerations hold for d_2 and d_4. Formally:*

$$\begin{array}{l} \alpha(d_1, a_1) = \textit{Italy} \\ \alpha(d_2, a_1) = \textit{Spain} \\ \alpha(d_3, a_1) = \textit{USA} \\ \alpha(d_4, a_1) = \textit{Italy} \\ \alpha(d_3, a_2) = \textit{12 Dec 2004} \\ \alpha(d_3, a_4) = \textit{1000\$} \end{array}$$

For all the other pairs document–attribute, α is undefined.

[6] This intuition will be formalized in Section 3.3.

Chapter 3

A semantic method for schema coordination

3.1 The matching problem

Schema matching is an operation relating certain elements of a schema S_1 to certain elements of a schema S_2. The semantics of the relations[1] depends on the particular instance of the problem, i.e. the cause for which schema matching is required. Generally, the mapping elements we expect to determine between two schemas of *classification* are relations between the content of the schema elements. Indeed, when dealing with *classification* schemas we are usually interested in the objects they contain and not in the schemas *per se*.

Consider the ACH of Figure 2.1, H_1, and the one of Figure 3.1, H_2. H_1 represents a classification structure which can be used, say, by a travel agency, whereas we may imagine H_2 to categorize the contract proposals of an insurance agency. The travel agency is not involved in the insurance business and it is not interested in managing policies directly. However, it may want to provide its clients with possible offers of insurance contracts for their holidays. Thus, in the context of a partnership between the two agencies, the former decides to import the policy offers of the latter. Obviously, the travel agency does not need all the possible contracts of the insurance agency but only those which deal with holiday events (in a certain country). The automatic identification of the parts of H_2 containing documents which are relevant for the travel agency and which can be reasonably imported in its classification categories can be viewed as a problem of *schema matching*. So, for example, we expect that the matching function is able to recognize that all the documents which are classified in the node `Italy` of H_2 can be also classified in the node `Insurance and lost baggage` of H_1, i.e. the relation between the node `Insurance and lost baggage` of H_1 and the node `Italy` of H_2 is *more general than*. In fact, we suppose that the former

[1] Rahm and Bernstein [40] discuss a set of possible expression values which have been used in literature.

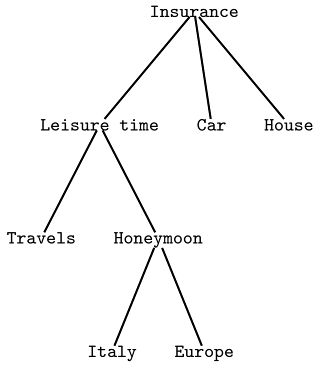


Figure 3.1: Classification structure for insurance policies

contains proposals of insurance policies for holidays in a given country[2], while the latter *contains* proposals of insurance policies for honeymoons spent in Italy. Given that a honeymoon is a vacation, we can derive that each contract for a honeymoon is also a possible contract for a vacation[3], or better, for a particular type of vacation.

We can formally define a *mapping* between two *hierarchical classifications with attributes* as in the following:

Definition 5 (Mapping) *A* mapping $\mathcal{M}_{H \to H'}$ *between two ACH,* $H = \langle K, E, \mathrm{att}, \mathrm{lab} \rangle$ *and* $H' = \langle K', E', \mathrm{att}', \mathrm{lab}' \rangle$*, is a set of* mapping elements $\langle k, k', r \rangle$*, where* $k \in K$*,* $k' \in K'$ *and* $r \in \{\subseteq, \supseteq, \equiv, \perp\}$*.*

A mapping is defined as a set of *set–theoretical* relations[4] between the nodes of two ACH structures. Our claim is that the type of relations we are interested in depends critically on the use of the structure. Because of the fact that ACH users are concerned with content classification (e.g. objects, documents) and *not* with abstract schemas, and that they are primarily interested in finding objects/documents, we derive that the relevant relations are the set–theoretical ones. We refer to these kinds of mappings as *pragmatic relations* since they are relations between the content of schema elements.

Literature does not propose a uniform and general criterium for evaluating the *soundness* of a mapping between two classification schemas. Thus, we introduce a notion of soundness which can be assumed as guideline in the performance evaluation of matching methods of classification schemas.

[2] Note that when a node label contains the 'and' conjunction (e.g. 'insurance and lost baggage'), we usually mean that, in the node, we can find elements about both the subjects ('insurance' and 'lost baggage') mentioned in the label. Thus, from a logical point of view, the 'and' conjunction corresponds to the *or* operator.

[3] But not vice-versa: not all the contracts related to vacations can be tailored to honeymoons.

[4] $m \perp n$ must be interpreted as $m \cap n = \emptyset$, where m and n are two sets.

In the following definition, we will use the notation $\mu_\tau(k)$ to indicate the set of objects classified in the node k of a given ACH H by a classification function $\langle\tau, \alpha\rangle$ of a set of objects D, i.e. $\mu_\tau(k) = \{d \in D \mid \tau(d) = k\}$. Moreover, we will use the notation $\mu_\tau(k{\downarrow})$ to denote the objects classified under the subtree rooted at k. Formally, let $k{\downarrow} = \{k' \in K \mid k' = k \textit{ or } k' \textit{ is a descendant of } k\}$ be the set of nodes in the subtree rooted at k, then $\mu_\tau(k{\downarrow}) = \bigcup_{k' \in k{\downarrow}} \mu_\tau(k')$.

Definition 6 (Soundness of a mapping) *Let $\langle\tau, \alpha\rangle$ and $\langle\tau', \alpha'\rangle$ be two classifications of a set of objects D in two ACH H and H', respectively. A mapping $\mathcal{M}_{H\to H'}$ is* sound *with respect to $\langle\tau, \alpha\rangle$ and $\langle\tau', \alpha'\rangle$ if the following conditions hold for each mapping element $\langle k, k', r\rangle \in \mathcal{M}_{H\to H'}$:*

- *if r is $\subseteq$ then $\mu_\tau(k) \subseteq \mu_{\tau'}(k'{\downarrow})$;*
- *if r is $\supseteq$ then $\mu_\tau(k{\downarrow}) \supseteq \mu_{\tau'}(k')$;*
- *if r is $\equiv$ then $\mu_\tau(k) \subseteq \mu_{\tau'}(k'{\downarrow})$ and $\mu_\tau(k{\downarrow}) \supseteq \mu_{\tau'}(k')$;*
- *if r is $\perp$ then $\mu_\tau(k) \cap \mu_{\tau'}(k') = \emptyset$.*

Mappings have to represent relations existing between sets of objects contained in the schemas. Consequently, a mapping is considered sound with respect to a classification if it identifies the correct set–theoretical relations between the elements of the structures we examine. Moreover, the definition considers the way objects are usually classified. Take the ACH H_1 rooted at `Vacations` and a single–node ACH, `Vacations`. The documents classified in the node `Last minute offers` of H_1 would be probably classified in the node `Vacations` of the single–node ACH. Thus, when considering, for example, the mapping element $\langle k, k', \subseteq\rangle$, we require that the set of objects in k is contained in k' and possibly in its descendants. Now, consider the relation of equality (i.e. $\equiv$) and a corresponding mapping element, $\langle k, k', \equiv\rangle$. In this case, we would be tempted to require the equality of the objects contained in the sub–trees rooted at the nodes k and k', but it would not be a good notion of soundness. Consider the two above mentioned ACH. An object which is classified under the node `Insurance and lost baggage` of H_1 would not be probably classified in any node of the single–node ACH. The cause resides in the fact that hierarchical classifications are not taxonomies where each object can be reasonably moved up in the hierarchy once the corresponding node has been removed. Dual considerations hold for the condition of soundness of a mapping element as $\langle k, k', \perp\rangle$. Consider, for example, the ACH H_1 and H_2 and the two corresponding root nodes. Probably, a node would not contain any object classified in the other one and vice versa, anyway we have already said that the two structures could share part of their content.

We can thus state that the goal of a matching problem is to find a sound mapping between two schemas possibly maximizing the number of the mapping elements identified.

3.2 Semantic methods

We have debated that mappings relating classification structures are meaningful if they have a pragmatic value. In order to find relations between sets of objects, we would need to check the instances contained in the two given schemas. Obviously, we could not tackle the problem exhaustively (i.e by considering all the possible objects of the universe which can be classified in the schemas) since it would result in an undecidable computation. However, in literature there are some approaches which use machine learning techniques to find mappings[5]. Machine learning techniques are based on induction since they essentially try to deduce the potential content of a node from a set of training examples[6]. The main drawback of these systems is that they require a relevant set of classified objects to work properly. Unfortunately, this is not always the case and in some situations we can also find schemas with no available content. Furthermore, note that considering the actual content of schema elements (without any mechanism of induction) would not generate good answers for a matching method. In fact, we could not discriminate between some set–theoretical relations. For example, if the nodes `Insurance and lost baggage` of H_1 and the node `Italy` of H_2 contained one document only concerning, say, 'insurance policies for honeymoons in Italy', we would derive that the two nodes are *equivalent*. But this result would have a contingent value only: it could not be true if we had some further documents. Similar cases occur when two nodes lying in very different schemas are both empty, and when two nodes lying in the identical position of identical schemas contain different sets of documents.

Therefore, we need to find an alternative way of tackling the matching problem. One possibility is to simulate the human way of acting. In order to analyze the human behaviour in case of a matching problem, we can simply look at the process which has led us to infer the relation *more general than* between the node `Insurance and lost baggage` and the node `Italy`. We have first associated a meaning to the two nodes and we have then derived the relation existing between such interpretations (we call the relation between the meaning of two nodes *semantic relation*). Finally, we have directly derived the pragmatic relation from the semantic one.

Actually, there are three basic (and implicit) assumptions which support and validate such derivation[7]:

1. our interpretation of the schemas is the same as the user (or the machine) which has classified the objects (i.e. the classifier);

2. all the objects of the universe which can be classified in the schemas are effectively classified under them;

3. objects are classified according to a precise criterium:

[5] See, for example, Doan, Madhavan et al. [25].

[6] Actually, they predict whether an instance d belongs to a category (node) k by training a classifier with positive and negative examples of instances belonging to k.

[7] Section 3.3 is devoted to formalize these aspects.

(a) each object d is associated to a meaning, $\phi(d)$ (i.e. its topic);

(b) each node k is associated to an interpretation, $\mathcal{I}(k)$ (i.e. the topic it classifies);

(c) each object is classified under a certain node if its topic is semantically related to the meaning of the node.

It is quite intuitive that relations between the meaning of the nodes do not match relations between their contents if our interpretation of the nodes is different from the one of the classifier (condition 1) or if objects are classified casually (condition 3). Indeed, if the meaning we have given to a node is different from the way the classifier intends it, we cannot expect that what we would classify in that node is the same as its actual content. Moreover, any deduction about the content of a node is tricky if the classifier puts a certain object in a node whose category does not represent the topic of the object. Finally, when considering semantic relations, we implicitly refer to the *potential* content of schema elements (condition 2) since we do not explore what they effectively contain.

Assuming the validity of such a process, *the problem of matching hierarchical classifications can be reduced to the problem of computing semantic relations between the meaning of their elements*. Our approach to the matching problem is based on this process, which we summarize in the following two steps:

Semantic explicitation: its goal is to build a formal representation of the meaning of the nodes of a given ACH. For example, it assigns a formal representation $\mathcal{I}(k)$ (i.e. 'insurance policies and lost baggage for holidays in a given country') to the node k, `Insurance and lost baggage` and a formal representation $\mathcal{I}(k')$ (i.e. 'insurance policies for honeymoons in Italy') to the node k', `Italy`.

Semantic comparison: its goal is to deduce the semantic relation holding between two nodes, i.e. the logical relation between their meanings. For example, it can derive that the interpretation $\mathcal{I}(k)$ of the node `Insurance and lost baggage` subsumes the interpretation $\mathcal{I}(k')$ of the node `Italy` according to some available knowldge $\mathcal{O}$, i.e. $\mathcal{O} \models \mathcal{I}(k') \sqsubseteq \mathcal{I}(k)$.

Our assumption is that we can then easily derive mappings from this analysis[8]. Benerecetti, Bouquet et al. [21] define *semantic methods* those which are characterized by the two steps mentioned.

In Algorithm 3.1, we present a top level representation of our semantic method of *schema matching*.

CtxMatch-2 takes two *hierarchical classifications with attributes* and produces a *mapping*. In steps 1 and 2, it builds the meaning of the elements of the two input schemas, H and H'. This results in two structures, H_s and H'_s, enriched with semantic information and with the interpretations of the

[8] Section 3.3 presents and formalizes the manner.

```
Algorithm 3.1 CtxMatch-2 ($H$,$H'$)
    ▷ ACH: $H = \langle K, E, \mathrm{att}, \mathrm{lab}\rangle,\ H' = \langle K', E', \mathrm{att}', \mathrm{lab}'\rangle$
    VarDeclarations
        semantic ACH: $H_s,\ H'_s$
        semantic relation: $r_s$
        pragmatic relation: $r_p$
        mapping: $\mathcal{M}_{H \to H'}$
1   $H_s \leftarrow$ SEMANTIC–EXPLICITATION ($H$);
2   $H'_s \leftarrow$ SEMANTIC–EXPLICITATION ($H'$);
3   for each pair of nodes $k \in K,\ k' \in K'$ do
4       $r_s \leftarrow$ SEMANTIC–COMPARISON ($H_s,\ H'_s,\ k,\ k'$);
5       $r_p \leftarrow$ CONVERSION ($r_s$);
6       $\mathcal{M}_{H \to H'} \leftarrow \mathcal{M}_{H \to H'} \cup \{\langle k, k', r_p\rangle\}$;
7   Return $\mathcal{M}_{H \to H'}$;
```

corresponding nodes. We call these structures *semantic ACH*[9]. In steps 4–6, CtxMatch-2 compares each pair of nodes of the two schemas and deduces the corresponding *semantic relation.* The semantic relation is then turned into a *pragmatic relation* (step 5) which is finally stored in the mapping variable $\mathcal{M}_{H \to H'}$ (step 6). The way *semantic relations* are turned into *pragmatic relations* is explained in the next section formally. Intuitively, each set–theoretical relation is simply mapped onto the corresponding logical relation (if the meaning of a node k subsumes the meaning of a node k' then CtxMatch-2 produces the mapping element $\langle k, k', \supseteq\rangle$, if the meaning of a node k is equal to the meaning of a node k' then it produces the mapping element $\langle k, k', \equiv\rangle$, etc.). Finally, step 7 returns the result of the process.

3.3 Formalization of semantic methods

In the previous section, we have provided the intuition of what a semantic method is and the way it works to discover mappings between schemas. We are now going to formalize the notions it is based on and the conditions which guarantee its soundness.

Initially, we formalize the concept of *topic* of a certain object d and of *interpretation* of a certain node k. Both definitions require the notion of *ontology.* Ontologies represent the knowledge of users about the world and the fundamental element of any semantic method. Indeed, they provide both the information used by the semantic explicitation process to interpret schema elements and the background knowledge needed to compare nodes semantically[10]. Essentially, ontologies are sets of assertions defined on real world concepts.

[9] They will be formalized in Section 6.1.

[10] We investigate these topics in the next chapter.

Definition 7 (Ontology) *An* ontology $\mathcal{O} = \langle \text{TBox}, \text{ABox} \rangle$ *is a knowledge base of a Description Logic, where* TBox *is a set of terminological axioms and* ABox *a set of factual axioms. They are defined on a set of concept names N_C, role names N_R and individual names N_I.*

In the following, we will refer to the set of all the possible concepts of a given ontology $\mathcal{O}$ as $\mathcal{C}$ and to the set of all the possible roles as $\mathcal{R}$.

We define the *topic* of an object d as the concept which best represents either the object itself or the subject it refers to (e.g. in case of documents, descriptions, images).

Definition 8 (Topic) *The* topic *of a set of objects D is a function $\phi_{\mathcal{O}} : D \to \mathcal{C}$ relating each object to a (complex) concept of a reference ontology $\mathcal{O}$.*

Note that an object may be associated to many different topics generally. For example, this thesis could be classified as 'semantic interoperability' or as 'ontology integration' or as 'schema matching', etc. Anyway, we require that only one subject is chosen, i.e. the most identifying.

We define as *interpretation* of an ACH node k the category it represents. It is identified by an ontological concept.

Definition 9 (Node interpretation) *The* interpretation *of a node k of an ACH $H = \langle K, E, \text{att}, \text{lab} \rangle$ is a function $\mathcal{I}_{\mathcal{O}} : K \to \mathcal{C}$ relating each node to a (complex) concept of a reference ontology $\mathcal{O}$.*

Note that there is not *the* right interpretation of a certain node but different interpretations can be equally reasonable in the reference context. For example, a user could refer to the node `Insurance and lost baggage` of H_1 as 'insurance policies for vacations', while another user as 'insurance policies stipulated on vacation'. A user could interpret the node `Vacations` (of the same ACH) as 'the time devoted to rest or pleasure', while another user as 'the act of making something legally void', etc.

We also use the notion of *interpretation* to denote the meaning of a non–empty string[11]. A string can be interpreted either as a concept or as a role of a certain ontology.

Definition 10 (String interpretation) *The* interpretation *of a string $s \in \mathcal{NL}$ is a function $\mathcal{I}_{\mathcal{O}} : \mathcal{NL} \to \{\mathcal{C} \cup \mathcal{R}\}$ relating each non–empty string either to a (complex) concept or to a (complex) role of a reference ontology $\mathcal{O}$.*

Note that both in the case of the *topic* and of the *interpretation*, there is no way to prevent dummy assignments, i.e. we are not able to guarantee that the topic we associate to an object is reasonable or that a schema element or a string are interpreted correctly. The only possible condition we can impose

[11] We use the same notation since the function we are referring to is determined by the corresponding domain. Hence, no ambiguity is possible.

is that they are validated by a human being. However, the problem is quite general and common to semantic theories based on formal logic[12].

Finally, we introduce the notion of *well–formed* classification. In the previous section, we have stated that objects must be classified according to a precise criterium in order to support and validate the implicit derivation of pragmatic relations from semantic ones (see condition 3 in the previous section). In the following definition, we are going to formalize such conditions which represent the usual human behaviour when classifying objects into an ACH.

Definition 11 (Well–formed classification) *A classification $\langle\tau, \alpha\rangle$ of a set of objects D in an ACH $H = \langle K, E, \mathrm{att}, \mathrm{lab}\rangle$ is* well–formed *with respect to an ontology $\mathcal{O}$, to an interpretation $\mathcal{I}_{\mathcal{O}}$ and to a topic $\phi_{\mathcal{O}}$ if the following conditions hold for each $d \in D$, for each $k \in K$, for each $a = \langle id, F\rangle \in A$ and for each $s \in \mathcal{NL}$:*

- *if $\alpha(d, a) = s$ and $F = F_r$ then $\mathcal{I}_{\mathcal{O}}(s) \sqsubseteq \mathcal{I}_{\mathcal{O}}(F_r)$;*
- *if $\alpha(d, a) = s$ then $\mathcal{O} \models \phi_{\mathcal{O}}(d) \sqsubseteq \exists\mathcal{I}_{\mathcal{O}}(id).\mathcal{I}_{\mathcal{O}}(s)$;*
- *if $k = \tau(d)$ then $\mathcal{O} \models \phi_{\mathcal{O}}(d) \sqsubseteq \mathcal{I}_{\mathcal{O}}(k)$;*
- *if $k = \tau(d)$ then there is no $k' \in K$ such that $\mathcal{O} \models \phi_{\mathcal{O}}(d) \sqsubseteq \mathcal{I}_{\mathcal{O}}(k')$ and $\mathcal{O} \models \mathcal{I}_{\mathcal{O}}(k') \sqsubseteq \mathcal{I}_{\mathcal{O}}(k)$;*
- *if there is a node $k' \in K$ such that $\mathcal{O} \models \phi_{\mathcal{O}}(d) \sqsubseteq \mathcal{I}_{\mathcal{O}}(k')$ then d is classified in H.*

The first two conditions deal with attributes. The former asserts that if an attribute is specified in terms of its *filler type*, the corresponding value must represent one of its possible instances. The condition prevents from assigning attribute values casually. So, we require that the interpretation of a value of the attribute a_1 in Example 1 is a Country (e.g. Italy, USA, France) intended as concept of a certain ontology. The latter guarantees that the topic of each object specifies a value for each attribute characterizing it. For example, the topic of the document d_1 in Example 3 must express the fact that the holiday it refers to will be spent in Italy. The third condition prevents from classifying documents blindly. It is stated that the topic of an object must be subsumed by the interpretation of the node where it is classified. It prevents, for example, from classifying an image of a landscape in the node `Insurance and lost baggage`. The fourth condition represents the *principle of specificity*: an object must be classified in the node whose interpretation is the most specific among those subsuming the topic of the object. Note that in *taxonomies*, this condition implies that objects are pushed down in the structure. In fact, a strong assumption about taxonomies is that objects under a certain node can be classified also under its ancestors, i.e. the ancestors of each node subsume its meaning. Unfortunately, hierarchical classifications do not have this property. For example,

[12] See the model–theoretic argument discussed by the philosopher H.Putnam [39] and the discourse in Benerecetti, Bouquet et al. [21].

consider the nodes `Insurance and lost baggage` and `Reports` of H_1. They classify 'insurance policies for holidays' and 'reports of spent vacations' respectively, but Insurance and Report are not Vacation which is the interpretation of their parent: they are simply *related* to Vacation. Finally, the fifth condition is quite trivial: it simply states that all the objects which can be potentially classified in the structure must be effectively classified in it. This condition is necessary since τ in a classification is partial, thus allowing objects not to be put in a structure.

Once we have defined the basic notions of a semantic method and we have determined the conditions which make a classification well-formed, it is quite intuitive to formalize the soundness of a semantic method. Note that we have already addressed the topic in the previous section by providing the intuitive conditions which allow to derive mappings from semantic relations correctly.

We use the notion of node interpretation with respect to two ACH H and H' as $\mathcal{I}_{\mathcal{O}} : \{K \cup K'\} \rightarrow \mathcal{C}$, where $K \in H$ and $K' \in H'$. Moreover, we use the notion of well–formed classification $\langle\langle\tau, \tau'\rangle, \langle\alpha, \alpha'\rangle\rangle$ of a set of objects D in two ACH H and H' with respect to an ontology $\mathcal{O}$, to a node interpretation $\mathcal{I}_{\mathcal{O}}$ (with respect to H and H') and to a topic $\phi_{\mathcal{O}}$, where $\langle\tau, \alpha\rangle$ is a well–formed classification of D in H and $\langle\tau', \alpha'\rangle$ is a well–formed classification of D in H'.

Proposition 1 *Let ω be a semantic method based on an ontology $\mathcal{O}_\omega$ and on the interpretation $\mathcal{I}_{\mathcal{O}_\omega}$ of two ACH H and H'. ω is* sound *with respect to a* well–formed *classification $\langle\langle\tau, \tau'\rangle, \langle\alpha, \alpha'\rangle\rangle$ of the whole* universe *of objects D in H and H' (with respect to $\mathcal{O}$ and $\mathcal{I}_{\mathcal{O}}$), if the following conditions hold:*

- *$\mathcal{O}_\omega \sqsubseteq \mathcal{O}$;*
- *for each $k \in H$ and $k' \in H'$, $\mathcal{I}_{\mathcal{O}}(k) = \mathcal{I}_{\mathcal{O}_\omega}(k)$ and $\mathcal{I}_{\mathcal{O}}(k') = \mathcal{I}_{\mathcal{O}_\omega}(k')$;*
- *ω derives mapping elements from semantic relations according to Table 3.1.*

Semantic relation	**Mapping element**
$\mathcal{O}_\omega \models \mathcal{I}_{\mathcal{O}_\omega}(k) \sqsubseteq \mathcal{I}_{\mathcal{O}_\omega}(k')$	$\langle k, k', \subseteq\rangle$
$\mathcal{O}_\omega \models \mathcal{I}_{\mathcal{O}_\omega}(k) \sqsupseteq \mathcal{I}_{\mathcal{O}_\omega}(k')$	$\langle k, k', \supseteq\rangle$
$\mathcal{O}_\omega \models \mathcal{I}_{\mathcal{O}_\omega}(k) \equiv \mathcal{I}_{\mathcal{O}_\omega}(k')$	$\langle k, k', \equiv\rangle$
$\mathcal{O}_\omega \models \mathcal{I}_{\mathcal{O}_\omega}(k) \sqcap \mathcal{I}_{\mathcal{O}_\omega}(k') \sqsubseteq \bot$	$\langle k, k', \bot\rangle$

Table 3.1: Mapping from semantic to pragmatic relations

The basic assumptions for a semantic method of schema matching to be sound (i.e. the mapping it returns is sound) are that the classification are well–formed (see condition 3 in the previous section) and the objects they consider represent the whole universe (see condition 2 in the previous section). These conditions have been already debated in depth. More attention is required for the ones we have itemized in the above proposition. First, the knowledge base used by the semantic method must be subsumed by the one used by the classifier. In fact, each inference process performed by the semantic method must be consistent with the reasoning of the classifier. Then, it is required that the schema interpretation of the classifier is the same as the semantic method (see condition 1 in the previous section). Finally, the semantic method must derive mapping elements from semantic relations according to the *conversion table* proposed. It is quite trivial since it associates the corresponding set–theoretical to each logical relation. Note that this transformation represents the mechanism used by the function CONVERSION of Algorithm 3.1 to translate semantic relations into pragmatic ones.

The major problem resides in the necessity to interpret the schema elements in the same way as a human classifier. The most we can do is trying to simulate the human behaviour and its capacity of overcoming the lack of semantics of schemas by means of its knowledge and experience. Anyway, the process must be led by the awareness of the non–existence of a solution but of the need of guessing and misinterpreting as every other user does.

Chapter 4

The methodology

4.1 Semantic explicitation

4.1.1 The problem

The first step in matching schemas with semantic methods is to reveal the implicit meaning of schema elements. Schemas do not completely express the semantics of the data they describe and much is left to the ability of the user to exploit the extant information in order to build the actual meaning of the nodes. We aim at deriving the meaning of the nodes of an ACH, in the same way as a human being interprets the labels of the structure. In particular, we propose a methodology to automatically build the concepts approximating the implicit node meaning. In this thesis, we discuss the possibility of using a very powerful logic framework as the Description Logic [20, 29, 19]. We express the meaning of each node as a concept term of a Description Logic defined on the set of concepts and roles of a certain ontology. We use the $\mathcal{ALCO}$[1] fragment of the DL for what concerns concept constructors, while the expressiveness of the roles is limited by the necessity of dealing with a decidable and manageable language.

Consider, for example, the ACH of Figure 2.1. In Section 2.1, we presented our intuitive interpretation of the meaning of its nodes, i.e. of the category that each element in the structure represents. The goal of semantic explicitation is both to try to achieve the same results of our reasoning and to express them in DL. Figure 4.1 shows the desired outcome of the process with reference to such a schema.

Each node k is associated to a concept term, $\mathcal{I}_{\mathcal{O}}(k)$, expressing the *node interpretation*. Concept names, role names and nominals used in the formulae are assumed to belong to the reference ontology $\mathcal{O}$. Each of them represents the *interpretation* of the words in the ACH labels and attributes. For example, Vacation is the *interpretation* of the string 'vacation' in the label of k_0, location

[1] $\mathcal{O}$ allows named individuals, called *nominals* [29], to occur in concepts.

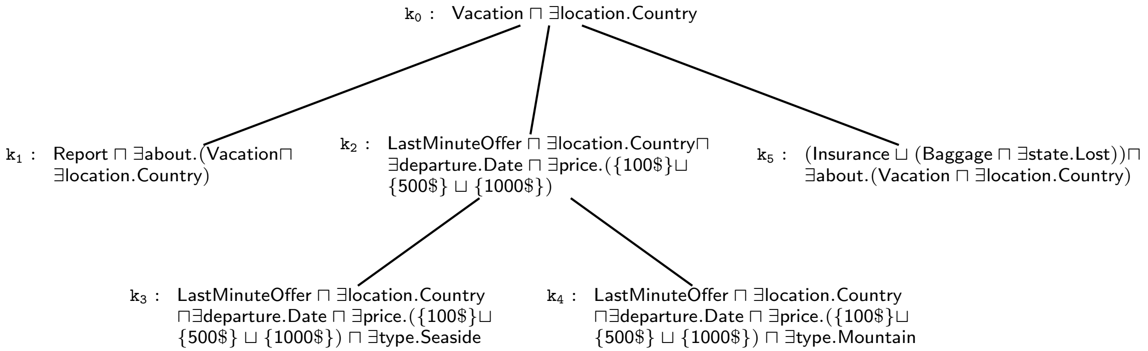

k₀ : Vacation ⊓ ∃location.Country
k₁ : Report ⊓ ∃about.(Vacation⊓ ∃location.Country)
k₂ : LastMinuteOffer ⊓ ∃location.Country⊓ ∃departure.Date ⊓ ∃price.({100$}⊔ {500$} ⊔ {1000$})
k₅ : (Insurance ⊔ (Baggage ⊓ ∃state.Lost))⊓ ∃about.(Vacation ⊓ ∃location.Country)
k₃ : LastMinuteOffer ⊓ ∃location.Country ⊓∃departure.Date ⊓ ∃price.({100$}⊔ {500$} ⊔ {1000$}) ⊓ ∃type.Seaside
k₄ : LastMinuteOffer ⊓ ∃location.Country ⊓∃departure.Date ⊓ ∃price.({100$}⊔ {500$} ⊔ {1000$}) ⊓ ∃type.Mountain

Figure 4.1: The expected result of the semantic explicitation process

of 'location' in the name of the attribute *location* of k_0, etc. Roles represent the implicit relations between the concepts of the structure. For example, Report and Vacation are assumed to be related by the role about (representing the subject of the report), LastMinuteOffer (here intended as 'last minute offer of vacation') and Seaside by type (representing the type of vacation, e.g. vacation at the seaside, on mountain), etc. Concept names, role names and nominals are assembled by means of the concept constructors of conjunction, disjunction and existential restriction. For example, k_0 respresents those instances which belong to the concept Vacation and which have a location–successor of the type Country, k_1 represents instances of Report which have an about–successor of the type of the concept represented by the root node, etc.

4.1.2 Knowledge for semantic explicitation

In order to make the meaning of schema nodes explicit, we need to consider different sources of information. The meaning of a node depends on its label and on its attributes but also on its position in the classification hierarchy. We refer to the information given by the node label and by its attributes as *local knowledge*, while we consider the information provided by its vicinities as *contextual knowledge*. The analysis of node labels, which are natural language strings, requires *grammatical knowledge* and *lexical knowledge* able to associate words with their possible meanings. Furthermore, we need *ontological knowledge* to connect semantically the different concepts which are involved in the meaning of the node in exam. The ontological knowledge is also the main support to disambiguate the possible meanings of a node and select the interpretation which is the most likely in the context of the given structure. The next sub–sections detail these sources of information.

Note that the semantic interpretation of a schema strongly depends on the knowledge used. Users having different perceptions and models of the world may interpret the same structure (or the same natural language sentence) in different ways. Furthermore, even if each user interacting with the ACH had the same knowledge, it would not be guaranteed that the respective interpretations are equal (see Section 3.3).

Local and Contextual knowledge

Let us consider the node `Insurance and lost baggage` of Figure 2.1 and the corresponding intuitive interpretation (see Section 2.1). *Local knowledge*, i.e. the node label, informs us that we are dealing with 'insurance policies'. On the other hand, the fact that `Insurance and lost baggage` is a child of `Vacations` makes us aware that we are not dealing with 'life policies' or 'car policies' but with 'insurance policies related to vacations'.

The information which is provided by the vicinities of the node in exam and which is involved in its meaning is called *contextual knowledge*. Our methodology considers the ACH path from the root node to k as the context of a given node k. We assume that all the information which is needed to interpret k can be found in that part of the ACH. Formally, a node context can be defined as follows:

Definition 12 (Node context) *Let $H = \langle K, E, \mathrm{att}, \mathrm{lab}\rangle$ be an ACH and $k \in K$ one of its nodes. The* context *of k is the ACH $H_k = \langle K', E', \mathrm{att}', \mathrm{lab}'\rangle$, where $K' \subseteq K$ contains exactly k and its ancestors, and all the other elements are the restriction of the corresponding component of H on K'.*

However, we may also draw more information from a larger context. For example, 'vacations in a given country' does not represent the most precise interpretation of the node `Vacations` of Figure 2.1. Actually, the presence of the child node `Last minute offers` affects its meaning: we could also interpret it as 'vacations in a given country which are not last minute offers'. This is due to the principle of specificity discussed in Section 3.3 that pushes last minute offers, which are also vacation offers, down in the hierarchical classification. Anyway, note that we are not wrong with the first interpretation: we are simply less precise. Other cases where out–of–context nodes may influence the meaning of a given node can be considered. Imagine, for example, the ACH in Figure 4.2.

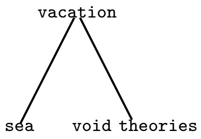


Figure 4.2: Context–tricky ACH

If we focused on the node `sea` only, we would be probably tempted to consider 'vacation' in its main meaning: 'leisure time away from work devoted to rest or pleasure'. However, this interpretation disagrees with the meaning of the node `void theories` that would probably intend 'vacation' as 'the act of making something legally void'. Thus, it is not clear whether we need either to preserve both meanings of 'vacation' or to focus on the meaning of 'vacation' in the given *context* only.

We may solve the problem by considering different notions of context of a given node. In Bouquet, Serafini et al. [23], the context of a node k was intuitively defined as the nodes lying in the path from the root to k and all the correponding children. Other notions can be imagined, till the extreme case when the whole hierarchical structure is considered. For the sake of simplicity, in this thesis we focus on the simplest notion of node context as in Definition 12.

Ontological knowledge

We have said so far that the context of a node contributes to its meaning and that the context is given by the nodes on its path. Unfortunately, such nodes are not semantically related in hierarchical classifications (and in schemas, in general). Indeed, the ACH edges does not represent relations between concepts, but simply relations between elements of the structure. However, if we want to interpret a certain node, we need to link the concepts represented by the nodes in its context. For example, a user could refer to the node `Insurance and lost baggage` of Figure 2.1 as 'insurance policies about vacations', while another as 'insurance policies stipulated on vacation'. In the former case the nodes `Vacations` and `Insurance and lost baggage` would be related by means of *about*, whereas in the second case by means of *stipulated on*. Each user will select the most appropriate and probable connection according to its own sensibility. The user's sensibility is mainly given by its experience and perception of the world. Such sensibility (at least the rational one) can be considered as the knowledge of the user about the world. We refer to this knowledge as *ontological knowledge*, and it can be formalized with ontologies (see Definition 7).

We define the ability of a user (or machine) to choose the most appropriate and probable semantic connection between two concepts according to its own knowledge as follows:

Definition 13 (Semantic Relation) *Let $\mathcal{SR}_{\mathcal{O}} : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{R}^*$ be a partial function from two concept terms of an ontology $\mathcal{O}$ to a possibly empty sequence of role terms, $\boldsymbol{R} = \langle \mathsf{R}_1, \mathsf{R}_2, \ldots, \mathsf{R}_n \rangle$ of the same ontology. $\boldsymbol{R}$ represents a* semantic relation *between the two input concepts.*

The function $\mathcal{SR}_{\mathcal{O}}$ takes two concepts as input and returns a sequence of roles. Intuitively, the output is the sequence of relations which is needed to go from the first input concept to the second by means of conceptual reasoning. In the previous examples, the concepts Insurance and Vacation were connected by the role about or stipulated on (i.e. **R** is a single–element set). If we considered the concepts Insurance and Seaside, we could say that the corresponding *semantic relation* is the sequence composed by the elements about and spending. In fact, we can think that Insurance is about-related to Vacations which in turn is spent-related to Seaside.

The function is not defined for a given input if the reference ontology cannot provide a reasonable relation between the two terms. When it returns an empty set, we mean that the former input concept is subsumed by the latter, i.e. they are *IsA*-related.

In Chapter 5, we formalize the concept of *semantic relation* and we define the function $\mathcal{SR}_{\mathcal{O}}$ from an algorithmic point of view.

Lexical knowledge

Local and contextual knowledge deal with labels and attributes which are natural language strings (i.e. words). On the other side, ontological knowledge deals with concepts. Now we need a bridge relating the space of words to the space of ontological concepts and roles. Such knowledge is called *lexical knowledge* and it represents the ability to associate a word[2] with its possible meanings (i.e. concepts). We model lexical knowledge as a *lexicon*, which we formally define as follows:

Definition 14 (Lexicon) *The* lexicon $\mathcal{L}_{\mathcal{O}} : \text{Lemma} \times \text{POS} \rightarrow 2^{\mathcal{C}}$ *is a function relating each lemma (with its part of speech) to a set of (atomic) concepts of a given ontology* $\mathcal{O}$*.*

A *lexicon* is a function from lemmas (and the corresponding parts of speech) to set of concepts, where lemmas and parts of speech have to be intended as in the NLP domain[3]. We assume to be able to recognize the lemma and the part of speech of a given word. Note that both the lemma and the part of speech are context–dependent, i.e they depend on the label and on the position in the structure where they occur.

Example 4 *Suppose that we have the string 'sea vacations' and that* WordNet[4] *is used as lexicon. Moreover, suppose we are interested in the lexical knowledge concerning the word 'vacations', whose context is given by the word 'sea'. We can easily derive that the lemma of 'vacations' is 'vacation' and that its part of speech in the given context is* noun*. The output of the* lexicon *function is* $\{\texttt{Vacation\#1}, \texttt{Vacation\#2}\}$*, where the elements of the set are the following concepts:*

1. `Vacation#1`*: leisure time away from work devoted to rest or pleasure;*

2. `Vacation#2`*: the act of making something legally void.*

The ability to identify the actual meaning of a word in a given context is a problem which is commonly referred to as *word sense disambiguation*[5] in the NL community. In the following, we will present our methodology of word sense disambiguation which is based on the following two observations: (i) we identify in the concatenation of the two words 'sea' and 'vacations' the concept 'holidays spent at the seaside' and (ii) we do not recognize any reason for associating the concept `Vacation#2` with the possible meanings of 'sea' (the context of the word 'vacations'). Note that it is our ontological knowledge (actually, *semantic relations*) which allows us to make such kind of reasoning.

[2] Note that in natural language, we can also find multi–words, i.e. words composed by multiple words to be considered as a single one. Some examples are: 'north America', 'last minute tour', etc.

[3] Lemmas are the 'standard' entries of a dictionary. For example, in English, we have the singular form for nouns (so that, for example, 'girls' and 'girl' refer to the same lemma: 'girl'), the infinite form for verbs (so that, for example, 'going' and 'went' refer to the same lemma: 'to go'), etc.

[4] In all the examples in the thesis, we have used WordNet as *lexicon.* WordNet synsets represent ontological concepts.

[5] See Chapter 9 for a survey of the major approaches.

Grammatical knowledge

ACH labels and attributes are usually composed by more than one word[6] (or expression). The analysis of the concept they represent depends on the ability to identify their atomic parts (as the words composing the statement), and, for each atomic element, its part of speech (i.e. noun, adjective, adverb, verb. etc.), its syntactical category (i.e. the dependency occurring with the other elements, e.g. it can be the subject of the phrase or the modifier of some subject) and its lexicon. The elements have to be subsequently connected semantically in order to derive the complex concept they represent. In other words, we need to identify the *grammatical* structure of a phrase. This is a common problem in NLP techniques and it is usually solved by building a *parse trees.* Formally, we define a parse tree as in the following:

Definition 15 (Parse Tree) *A* parse tree $P_{\mathcal{L}} = \langle N, G \rangle$ *is a finite rooted tree. Nodes are defined as 5-tuples* $\langle \text{Word}, \text{Lemma}, \text{SynCat}, \text{POS}, \text{Senses} \rangle$ *or as the symbol* @*.* Word *is an expression in* $\mathcal{NL}$ *whose lemma,* Lemma*, is an entry of a* lexicon $\mathcal{L}_{\mathcal{O}}$ $(= \mathcal{L})$*.* SynCat *and* POS *are respectively the syntactical category and the part of speech of* Word *in the given context, and* Senses *is the output of* lexicon *applied to* Lemma *and* SynCat*,* $\text{Senses} = \mathcal{L}_{\mathcal{O}}(\text{Lemma}, \text{SynCat})$*.*

Parse trees are structures that point out the dependencies between the elements of a phrase or a string. Such elements are represented by the parse tree nodes and are single– or multi–words. Each element is associated with some grammatical information and with its possible meanings.

Example 5 *Let us consider the phrase 'sea vacations' of Example 4. We can identify two words: 'vacations' and 'sea'. The respective lemmas are 'vacation' and 'sea'. Due to the grammatical structure of the English/American language, 'vacation' is a noun (part of speech) and it represents the phrase subject (syntactical category), whereas 'sea' is an adjective (part of speech) which qualifies the subject (i.e. it is the modifier). The corresponding parse tree is represented in Figure 4.3. The node which refers to 'sea' is classified as child of 'vacation'. By accessing the* lexicon*, it is possible to consider the possible word meanings: 'vacation' can denote the concepts* `Vacation#1` *and* `Vacation#2`*, while 'sea' can denote the concept* `Sea#1`*.*

Parse trees will be further investigated in Section 6.2.

4.1.3 Complete example

We are now going to present a complete example which starts from a node of a schema and creates the corresponding concept term. In particular, we will explore the different phases of the process by showing what steps human beings

[6] In case of compact notations, we can also find more words appearing as a single non–spaced word (e.g. 'BomberPlane' can be decomposed in 'Bomber' and 'Plane', 'Public_Company' in 'Public' and 'Company').

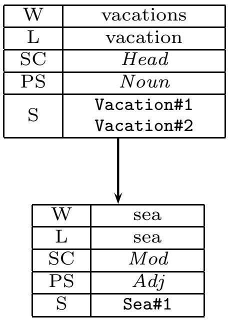


Figure 4.3: Parse tree of the label 'sea vacations'

implicitly do when interpreting classification structures. Our methodology is a simulation of such a process for what concerns the output of each step. The algorithms which lead to the intermediate results will be discussed in detail in Chapter 6.

The node we will analyze in the rest of the section is `Insurance and lost baggage` of Figure 2.1. Our goal is to use all the described sources of knowledge to derive its meaning. We expect that the result of the process is a concept term expressing that the node concerns 'insurance policies and lost baggage related to vacations spent in a given country'.

Firstly, we focus on the part of the ACH which contains information which we consider relevant for the interpretation of the node meaning, i.e. the node *context*. In this case, it is represented both by the node `Insurance and lost baggage` and by the root node, `Vacations`. Figure 4.4 shows the context of the node in exam.

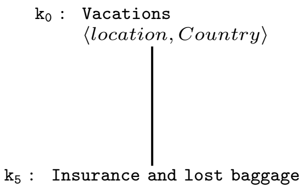


Figure 4.4: Context of the node `Insurance and lost baggage`

The next step analyzes the *local* meaning of each node in the given context by parsing the corresponding label and attributes. Figure 4.5 shows the parse tree of the node `Insurance and lost baggage`. We have identified two subjects, 'insurance' and 'baggage' (the symbol @ indicates that the corresponding children are not dependent, i.e. they are coordinate terms) and 'lost' as modifier of the latter.

Figure 4.6 shows the parse tree of `Vacations`. It has 'vacation' as root node and 'country' as modifier. The modifier represents the filler of the attribute

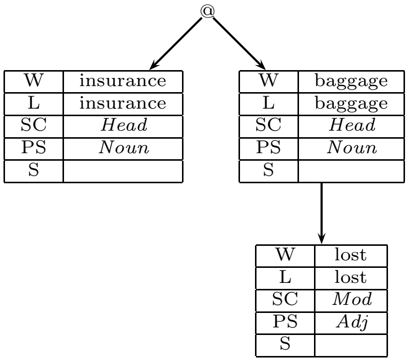


Figure 4.5: Parse tree of the label ‘insurance and lost baggage’

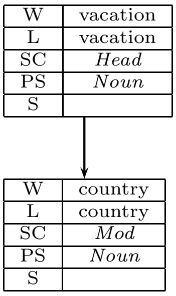


Figure 4.6: Grammatical structure of `Vacations`

location. A node attribute represents something that qualifies the concept expressed by that node. Therefore, we can consider its filler as a modifier of the label subject(s).

Note that the two structures identify the dependencies among the terms of the nodes but they do not highlight the type of semantic relation existing between them.

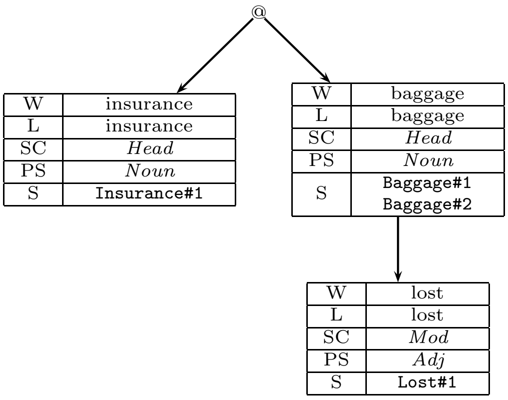
Words do not actually represent concepts. Thus, the next step is to exploit *lexical* knowledge to relate each word with its possible meanings. We have already discussed the two possible meanings of the word 'vacation' as noun, namely `Vacation#1` and `Vacation#2`. Moreover, we assume that 'country' and 'insurance' as nouns and 'lost' as adjective have one possible meaning, respectively:

1. `Country#1`: a particular geographical region of indefinite boundary;
2. `Insurance#1`: promise of reimbursement in the case of loss;
3. `Lost#1`: no longer in your possession or control.

Finally, we associate 'baggage' to two possible concepts:

1. `Baggage#1`: used to carry belongings when traveling;
2. `Baggage#2`: a worthless or immoral woman.

Note that the lemma and the part of speech of each word (which are required to access the *lexicon* function) are provided by *grammatical* competence according to the context in exam (i.e. the phrase which the word belongs to). The structure in Figure 4.7 represents the parse tree of the node `Insurance and lost baggage` enriched with concepts, while Figure 4.8 that of `Vacations`.



W	insurance
L	insurance
SC	*Head*
PS	*Noun*
S	`Insurance#1`

W	baggage
L	baggage
SC	*Head*
PS	*Noun*
S	`Baggage#1` `Baggage#2`

W	lost
L	lost
SC	*Mod*
PS	*Adj*
S	`Lost#1`

Figure 4.7: Parse tree of `Insurance and lost baggage` enriched with the lexicon

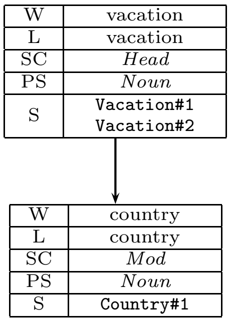


Figure 4.8: Parse tree of `Vacations` enriched with the lexicon

By means of this operation, we have considered all the possible meanings of each word. Actually, words do not usually represent all the concepts they can refer to but their meaning depends on the corresponding context. All of us would probably interpret 'vacation' as `Vacation#1` and 'baggage' as `Baggage#1` in the given ACH. In order to interpret with good precision the meaning of the node `Insurance and lost baggage`, we need to perform the same selection automatically. Given that the meaning of each word depends on its context, the corresponding disambiguation can occur by analyzing how each concept can be semantically linked to the rest of the context.

Thus, the next step in semantic explicitation is to search possible semantic relations existing between the concepts in the *context* (we call it *semantic enrichment*). *Ontological* knowledge supports us during this process. We assume that it provides us with the most likely semantic relation, if any, between two given concepts through the $\mathcal{SR}_{\mathcal{O}}$ function. Suppose it returns the following *semantic relations* composed by one role:

1. `Baggage#1` is related to `Lost#1` by means of `state#3`;
2. `Insurance#1` is related to `Vacation#1` by means of `about#1`;
3. `Vacation#1` is related to `Insurance#1` by means of `organizedBy#1`.

Note that Description Logic roles are returned as senses and not as strings which can be ambiguous. In the first case, `state#3` refers to 'the way something is with respect to its main attributes', in the second case, `about#1` refers to 'imprecise but fairly close to correct', and in the third case `organizedBy#1` refers to 'planned and directed'. In the case of attributes, the semantic relation between the node concept and the filler is already explicit in the form of a natural language string, i.e. the name of the attribute. Unfortunately, a string does not express a role (which must be associated to the intended meaning), therefore we need to look out for possible meanings of the attribute name which represent possible relations between one (or more) meanings of the subject of the node label and one (or more) meanings of the filler. In the example, we assume that ‘location’ can only refer to ‘a point or extent in space’, `location#1`, and that it represents a reasonable semantic connection between the concepts `Vacation#1` and `Country#1` only.

Now, all the semantic information derivable from the structure has been elicited. Hence, it is time to make precise choices about the meaning of each word and the way it is linked to the rest of the context. Note that such choices are the result of experience and reason, i.e. of heuristic criteria based on the available semantic information.

For example, we may discard the sense `Vacation#2` of ‘vacation’ because it is not related to any other concept in the context. On the contrary, the other meaning of ‘vacation’ is semantically connected both to `Country#1` by means of `location#1` and to `Insurance#1` by means of `about#1` and `organizedBy#1`. In other words, we are saying that ‘vacation’ as ‘the act of making something legally void’ is completely disjoint from the rest of the schema. Similarly, we may discard the concept `Baggage#2` because it is independent from the rest of the context, whereas `Baggage#1` is related to `Lost#1` by means of `state#3`. The result of this operation (we call it *semantic filtering*) is that we have associated each word with a more precise meaning. The criteria we have used to perform the process are very similar to the ones human beings implicitly use. In fact, we have considered the meaning which better fits in the given context as the most probable.

The last step before formalizing the meaning of the node in exam is to choose a set of semantic relations which allow a sensible interpretation. For example, the nodes `Vacations` and `Insurance and lost baggage` could be linked both as ‘insurance policies for holidays’ and as ‘vacations organized by insurance (agencies)’ according to the relations previously found. We could certainly say that the interpretation of the node `Insurance and lost baggage` is ‘insurance policies for holidays *or* vacations organized by insurance (agencies)’. Anyway, we would lack precision since the intended meaning is probably only one of those we have mentioned. Unfortunately, the choice is not easy and the criteria human beings use are based on their sensibility and on the evaluation of the coherence of the choice with respect to the rest of the context. For this example, suppose we prefer the first possible interpretation, thus discarding the relation `organizedBy#1` (and the concepts it links if they are not related to any other element in the context).

The process of semantic explicitation of the node `Insurance and lost baggage` ends by assembling the concepts of the context in such a way to represent the meaning of the node. Our formal language is the Description Logic framework defined on the set of concepts and roles which we have met during the process.

First of all, we must identify the concept term representing the local meaning of the nodes in the context. The corresponding grammatical structures suggest the dependency subject–modifier between the node elements, while the semantic relations identify the type of dependency. Intuitively, we formalize a dependency subject–modifier as a concept (i.e. the subject) with an existential restriction of the involved relation on the modifier[7].

With reference to the node Vacations, we thus express its interpretation as:

$$\texttt{Vacation\#1} \sqcap \exists \texttt{location\#1}.\texttt{Country\#1}$$

which represents the intuitive concept ‘holidays spent in a given country’.

[7] Section 6.6 futher investigates the way of formalizing a node meaning in DL.

Similarly, we interpret `Insurance and lost baggage` as:

$$\texttt{Insurance\#1} \sqcup (\texttt{Baggage\#1} \sqcap \exists\texttt{state\#3.Lost\#1})$$

representing 'insurance policies and lost baggage'.

Finally, we need to semantically link the concepts expressed by the two nodes. Note that in this case the ACH structure does not provide any information about which is the subject and which is the modifier. Thus, we must only rely on the semantic relations we have found between the two nodes, i.e. `about#1` between `Insurance#1` and `Vacation#1`. The composition of the interpretations of the two nodes results as follows:

$$\begin{array}{l}(\texttt{Insurance\#1} \sqcup (\texttt{Baggage\#1} \sqcap \exists\texttt{state\#3.Lost\#1}))\sqcap \\ \exists\texttt{about\#1}.(\texttt{Vacation\#1} \sqcap \exists\texttt{location\#1.Country\#1})\end{array}$$

It corresponds to the intuitive interpretation of the node `Insurance and lost baggage` as 'insurance policies and lost baggage about vacations in a country'. Note that the meaning of the root node represents the modifier of that of the child node. Finally, note that an attribute keeps the semantic connection with the node to which it is attached. In other words, *location* refers to 'vacations' independentely from the node we are examining.

4.1.4 The algorithm

In the previous sections, we have presented the information which is required to make the meaning of a schema explicit and the expected process which leads from the pure structure to the concepts it represents. In this section, we propose an algorithm to simulate this process. In particular, we define the steps which we have discussed in the previous section from an operative point of view.

The function SEMANTIC–EXPLICITATION in Algorithm 3.1 takes an ACH as input and returns a semantic ACH, H_s, containing the interpretation $\mathcal{I}(k)$ of each node k. The process adds and arranges grammatical and semantic information in H_s, gradually, till the final synthesis in the schema intepretation.

Algorithm 4.1 reproduces the process by clearly indicating the macro–steps of the procedure. Chapter 6 will be devoted to investigate the corresponding sub–steps.

For each node of the structure, the algorithm performs the steps we have presented in the previous section. We first focalize on the context of k (step 2). Then, we associate each node of the context to the corresponding parse tree (step 3), i.e. we add grammatical and lexical information to the input ACH. The resulting structure is a semantic ACH. In step 5 and 6, we complete the local analysis of the nodes by searching all the possible semantic relations which connect the different elements of a single node (step 5). Moreover, we discard some possible meanings (step 6) according to some local analysis. The steps we have presented so far are local processes because they act on each node independently, whereas steps 7 and 8 work on the basis of contextual considerations. Step 7 searches all the semantic relations between different

Algorithm 4.1 SEMANTIC–EXPLICITATION (H)

```
   ▷ ACH: $H = \langle K, E, \mathrm{att}, \mathrm{lab} \rangle$
   VarDeclarations
       context: $H_k$
       semantic $ACH$: $H_s$
1  for each $k \in K$ do
2      $H_k \leftarrow$ CONTEXT$(H, k)$;
3      $H_s \leftarrow$ PARSE–TREE$(H_k)$;
4      for each node $k' \in H_s$ do
5          $H_s \leftarrow$ LOCAL–SEMANTIC–ENRICHMENT$(H_s, k')$;
6          $H_s \leftarrow$ LOCAL–SEMANTIC–FILTERING$(H_s, k')$;
7      $H_s \leftarrow$GLOBAL–SEMANTIC–ENRICHMENT$(H_s)$;
8      $H_s \leftarrow$ GLOBAL–SEMANTIC–FILTERING$(H_s)$;
9      $H_s \leftarrow$ CHOOSE–COVERAGE$(H_s)$;
10     $\mathcal{I}(k) \leftarrow$ COMPOSING–THE–MEANING$(H_s)$;
11 Return $H_s$;
```

nodes by means of ontological knowledge. Step 8 performs the same filtering as step 6 but with a global perspective (i.e. with respect to the context of k). Indeed, it discards all those node meanings which are not related to the rest of the context. Note that step 6 and 8 jointly work by simulating the same reasoning that led us to discard the concepts `Vacation#2` and `Baggage#2`. Step 9 selects a set of semantic relations so that we can interpret k in a univocal way. It results in a filtering of the relations (and of the senses accordingly) in the semantic ACH. This process is the one which made us discard the relation `organizedBy#1`. Finally, step 10 builds the DL term expressing the meaning of the node k and it stores the result in the variable $\mathcal{I}(k)$ of the semantic ACH which will be finally returned to CTXMATCH-2. The expansion of the function COMPOSING–THE–MEANING in Section 6.6 will show that we first formalize the local interpretation of the contextual nodes and we subsequently assemble them in $\mathcal{I}(k)$.

Note that the algorithm is not optimized computationally. In fact, it must interpret all the nodes of the structure thus making some operations repetitive. In particular, steps 3–7 can be executed only once because they do not depend on the node we are analyzing or, in other words, they are not context dependent.

4.2 Semantic comparison

4.2.1 The problem

The second step in matching schemas with semantic methods is to derive relations between nodes (i.e. mappings) by the relations between the associated meanings. We named these relations *semantic relations* because they hold between concepts expressed in some formal language. In our case, a node meaning is a term in Description Logic. Thus, the relations we are interested in are the set of DL logical operators $\{\sqsubseteq, \sqsupseteq, \equiv\}$. Moreover, we use the notation $\perp$ to indicate that two concepts are disjoint, i.e. $\mathsf{C}\perp\mathsf{D}$ means that $\mathsf{C} \sqcap \mathsf{D} \sqsubseteq \perp$ (the empty concept). Then, each symbol will be mapped out on the set–theoretical relation $\subseteq$, $\supseteq$, $\equiv$ and $\perp$, respectively. According to Section 3.3, such a translation allows us to derive pragmatic relations from semantic ones and it provides the elements for a *mapping* $\mathcal{M}_{H \to H'}$ between two ACH structures H and H'.

Semantic comparison is based on the use of DL reasoners and it is supported by ontological knowledge[8] to make inference, as we show in the next section.

4.2.2 Complete example

Suppose we have to match the ACH of Figure 2.1 and the one of Figure 3.1 and that we are interested in the relation holding between the nodes `Insurance and lost baggage` and `Italy`, respectively.

We expect that the process of semantic explicitation has produced the following concept term, C, for the former node:

$$\begin{array}{l}(\texttt{Insurance\#1} \sqcup (\texttt{Baggage\#1} \sqcap \exists\texttt{state\#3.Lost\#1}))\sqcap \\ \exists\texttt{about\#1.}(\texttt{Vacation\#1} \sqcap \exists\texttt{location\#1.Country\#1})\end{array}$$

The most intuitive interpretation of the node `Italy` is 'insurance policies for honeymoons spent in Italy'. Assuming that the only meaning of 'honeymoon' and 'Italy' are respectively 'a holiday taken by a newly married couple', `Honeymoon#1`, and 'a republic in south Europe', `Italy#1`, we can formally express the meaning, D, of the node `Italy` as:

$$\texttt{Insurance\#1} \sqcap \exists\texttt{about\#1.}(\texttt{Honeymoon\#1} \sqcap \exists\texttt{location\#1.Italy\#1})$$

Moreover, suppose we have an ontology $\mathcal{O}$ containing the following axioms:

$$\begin{array}{l}\texttt{Honeymoon\#1} \sqsubseteq \texttt{Vacation\#1} \\ \texttt{Italy\#1} \sqsubseteq \texttt{Country\#1}\end{array}$$

Semantic comparison evaluates if it is possible to infer some relation between the meaning of the nodes according to the knowledge available. By means of a DL reasoner we can easily entail from $\mathcal{O}$ that $\mathsf{D} \sqsubseteq \mathsf{C}$, i.e.:

$$\mathcal{O} \models \mathsf{D} \sqsubseteq \mathsf{C}$$

We can conclude that the *semantic relation* between the meaning of the nodes `Insurance and lost baggage`, k, and `Italy`, k', is *more general than.* This result will finally produce the mapping element $\langle k, k', \supseteq\rangle$ according to the conversion table of Section 3.3.

[8] Note that the knowledge base used in semantic comparison is not necessarily the same as employed in semantic explicitation. Moreover, the reference ontologies used to interpret the two input schemas of the matching problem may be different, too. In the worst case, the different parties will be not able to communicate because they do not share the same concept and role names thus not finding any match.

4.2.3 The algorithm

The algorithm of semantic comparison uses the notion of logical consequence to derive the relation holding between two DL terms. The degree of DL expressivity used to formalize the ACH semantics strongly affects the algorithm efficiency and decidability. The crucial choice is represented by the complexity of the roles returned by the function $\mathcal{SR}_{\mathcal{O}}$ (see Definition 13). The decidability of the DL framework made of the $\mathcal{ALCO}$ fragment and the role constructors involved guarantees the same property to the semantic comparison algorithm, as well.

Algorithm 4.2 is that we apply to identify the *semantic relation* holding between two nodes k and k' of two ACH H and H'.

```
Algorithm 4.2 SEMANTIC–COMPARISON($H_s, H'_s, k, k'$)
    ▷ semantic ACH: $H_s$, $H'_s$
    ▷ node: $k \in H_s$, $k' \in H'_s$
    VarDeclarations
        reference ontology: $\mathcal{O}$
        semantic relation: $r$
1   if $\mathcal{O} \models (\mathcal{I}(k) \sqcap \mathcal{I}'(k')) \sqsubseteq \bot$ then $r \leftarrow \bot$;
2   else if $\mathcal{O} \models (\mathcal{I}(k) \equiv \mathcal{I}'(k'))$ then $r \leftarrow \equiv$;
3   else if $\mathcal{O} \models (\mathcal{I}(k) \sqsubseteq \mathcal{I}'(k'))$ then $r \leftarrow \subseteq$;
4   else if $\mathcal{O} \models (\mathcal{I}(k) \sqsupseteq \mathcal{I}'(k'))$ then $r \leftarrow \supseteq$;
5   else $r \leftarrow *$;
6   Return $r$;
```

The input of the algorithm is given by two semantic ACH and the two respective nodes we need to sematically compare. Steps 1–4 determine the logical relation existing between their concept terms, $\mathcal{I}(k)$ and $\mathcal{I}(k')$. The variable $\mathcal{O}$ represents the knowledge source we use to support the entailment process.

Note that if we do not find any relation between the concepts, we return the relation $*$ which we interpret as *possible intersection* or *compatibility.* Essentially, we want to have a placeholder for potential relations that we cannot deduce, but we cannot reject, either. For example, 'dogs' and 'cats' will be evaluated as compatible unless we have positive information that dogs and cats are disjoint. Of course, one might say that this conclusion is wrong, but we need to consider that in other cases (e.g. 'churches' and 'monuments') it would be right (for the sake of argument, let us imagine that there is no explicit connection between churches and monuments in the knowledge base in use). So the decision is between reasoning under a *closed world assumption*, which would prevent us from finding any relation in both cases, or reasoning under an *open world assumption*, which would allow us to find a relation of compatibility in both cases. The open world assumption increases the recall, but decreases the precision of the algorithm, whereas the closed world assumption has the opposite effect. Our decision was to go for an open world solution. In fact most concrete ontologies are not rich enough to support the conjecture that anything which cannot be deduced is false.

Chapter 5

Discovering semantic relations between concepts

5.1 Introduction

Semantic relations are a way to connect different concepts: given a certain context and a set of concepts, we identify the possible ways of connecting them and we choose the most appropriate in that case.

Consider, for example, the concepts ‘person’ and ‘name’. All of us would probably associate them by asserting that each person *has* a name, where ‘having a name’ represents the relation which occurs between the two concepts. However, one could also relate them by ‘having a nickname’ or ‘writing’, etc., and none of them may be considered the right one or wrong.

Consider also this further example: ‘person’ and ‘John’. In this case, we cannot state that each person’s name is John but we can guess that, if a relation between the concepts must occur, it would be probably of the type ‘has name’. In other words, we are saying that the name of a person is not John necessarily, but it could be. Anyway, even in this case, other possible relations may be considered.

Note that each user relates concepts according to its own knowledge, i.e. to its own model of the world and to some evaluations which are influenced by the context.

Ontologies contain the knowledge about semantic relations and the possible ways of connecting concepts. The fact that each person, Person, has a name, Name, may be formalized in an ontology $\mathcal{O}$ as:

$$\mathsf{Person} \sqsubseteq \exists \mathsf{hasName}.\mathsf{Name}$$

where Person and Name are in N_C and $\mathsf{hasName}$ in N_R. The axiom expresses that each person *has to* have a name. On the other hand, the fact that ‘John’ is a possible name for a person may be formalized as follows:

$$\begin{array}{l}\mathsf{Person} \sqsubseteq \exists \mathsf{hasName.Name} \\ \{\mathsf{John}\} \sqsubseteq \mathsf{Name}\end{array}$$

where the former axiom states that each person has a name, while the latter that John is a name. Actually, John is one *possible* name, since other concepts (or individuals) could be names too. In the following, we will refer to the first type of relations between concepts as *necessary*, while to the second type as *possible*. Our goal is to investigate a formal definition of semantic relation and a methodology to extract this kind of knowledge from ontologies. Section 5.2 formally defines what we mean for two concepts to be semantically related, while Section 5.3 presents a decidable algorithm to extract semantic relations from ontologies (i.e. the $\mathcal{SR}_{\mathcal{O}}$ function of Definition 13). Finally, Section 5.4 will discuss the formal differences between *possible*, *necessary* and *tautological* relations with respect to a certain ontology.

5.2 Semantically related concepts

We first introduce the notion of *simple* semantic relation and then we extend it to the one of *semantic relation*.

Before the definitions, we present a notation which we will use in the following in order to simplify the reading. Given an ontology $\mathcal{O} = \langle TBox, ABox \rangle$ and two concept terms C and D in $\mathcal{C}$, $\mathcal{O} \models \mathsf{C} \sqsubset \mathsf{D}$ means that:

- $\mathcal{O} \models \mathsf{C} \sqsubseteq \mathsf{D}$;
- $\mathcal{O} \not\models \mathsf{D} \sqsubseteq \mathsf{C}$.

The notation simply introduces the notion of strict subsumption.

Definition 16 (Simple semantic relation) *Given an ontology* $\mathcal{O} = \langle \mathrm{TBox}, \mathrm{ABox} \rangle$, *two concepts* C *and* D *in* $\mathcal{C}$ *are* semantically related *if there is a role* R *in* $\mathcal{R}$ *satisfying one of the following conditions:*

1. $\mathcal{O} \models \mathsf{C} \sqsubseteq \exists \mathsf{R.D}$*;*
2. *there is a concept* E *in* $\mathcal{C}$ *such that:*
   (a) $\mathcal{O} \models \mathsf{C} \sqsubseteq \exists \mathsf{R.E}$ *and* $\mathcal{O} \models \mathsf{D} \sqsubset \mathsf{E}$*;*
   (b) *there is* no *concept* F *in* $\mathcal{C}$ *such that:* $\mathcal{O} \models \mathsf{C} \sqsubseteq \exists \mathsf{R.F}$ *and* $\mathcal{O} \models \mathsf{F} \sqsubset \mathsf{E}$.

and $\mathsf{C} \sqcap \exists \mathsf{R.D}$ *is satisfiable with respect to* $\mathcal{O}$*, i.e.:*

$$\mathcal{O} \not\models \mathsf{C} \sqcap \exists \mathsf{R.D} \sqsubseteq \bot$$

R *represents a* simple semantic relation *between concepts* C *and* D.

The first condition refers to the definition of *necessary* relation as presented intuitively in the introduction. It states that C is semantically related to D by means of R, if each instance of C is required to have an R-successor which is an instance of D. The second condition corresponds to *possible* relations. It requires the existence of a concept E with the same property of D in the first condition and which subsumes (strictly) the target concept D. Condition 2.b allows to prevent from some tricky cases which can emerge when expressing *possible* relations in DL. Indeed, we must guarantee that the relation between C and E is directly asserted in the ontology. Consider, for example, an ontology composed of the following axioms:

$$\begin{aligned}&\mathsf{Sunset} \sqsubseteq \exists\mathsf{hasColor}.\{\mathsf{Red}\}\\&\{\mathsf{Red}\} \sqsubseteq \mathsf{Color}\\&\{\mathsf{Green}\} \sqsubseteq \mathsf{Color}\end{aligned}$$

Condition 2.a would find a *simple* semantic relation of the type $\mathsf{hasColor}$ between Sunset and $\{\mathsf{Green}\}$, which does not match what we intend for semantic relation. Indeed, if the ontology considered the possibility for a sunset to be green, it would assert that (assuming that no other colors are present in the ontology):

$$\mathsf{Sunset} \sqsubseteq \exists\mathsf{hasColor}.\mathsf{Color}$$

Instead, by linking Sunset with $\{\mathsf{Red}\}$ only, it is explicitly excluding the case that a sunset is of a color different from red. Thus, $\mathsf{hasColor}$ cannot be considered a possible relation between Sunset and $\{\mathsf{Green}\}$.

Finally, note that an ontology could also return no simple semantic relations between two concepts or it could return more relations (i.e. there are more roles satisfying the conditions of Definition 16).

Example 6 *Consider an ontology* $\mathcal{O} = \langle \mathrm{TBox}, \mathrm{ABox} \rangle$ *defined on the set of concept and role names of Table 5.1 and suppose the* TBox *to be composed as in Table 5.2.*

N_C	N_R
Insurance	about
Vacation	spendingPlace
Honeymoon	departure
Location	
Leisure	
Date	

Table 5.1: Concept and role names

According to Definition 16, we can identify at least 7 simple semantic relations between atomic concepts. We represent the pairs which are related by means of the about *role in the left–hand part of Table 5.3, those which are related by means of the* $\mathsf{spendingPlace}$ *role in the center and those related by means of the* $\mathsf{departure}$ *role in the right–hand part.*

Axioms
$\mathsf{Insurance} \sqsubseteq \exists \mathsf{about}.\mathsf{Vacation}$
$\mathsf{Vacation} \sqsubseteq \mathsf{Leisure}$
$\mathsf{Honeymoon} \sqsubseteq \mathsf{Vacation}$
$\mathsf{Leisure} \sqsubseteq \exists \mathsf{spendingPlace}.\mathsf{Location}$
$\mathsf{Honeymoon} \sqsubseteq \exists \mathsf{departure}.\mathsf{Date}$

Table 5.2: *TBox* axioms

about	spendingPlace	departure
(1) Insurance → Vacation	(1) Leisure → Location	(1) Honeymoon → Date
(2) Insurance → Leisure	(2) Vacation → Location	
(3) Insurance → Honeymoon	(3) Honeymoon → Location	

Table 5.3: *Simple* semantic relations between atomic concepts

All the semantic relations, except for the number 3 of the about *role, are given by the first condition of Definition 16. In fact, we can entail them from the given ontology. On the contrary,* Insurance *and* Honeymoon *are* about*–related for condition 2 of Definition 16 because* Honeymoon *is subsumed by* Vacation *which is an* about*-successor of the* Insurance *concept.*

Actually, we could identify some more simple semantic relations if we considered complex roles. For example, Insurance *and* Location *are semantically related by the role term:*

$$\mathsf{about} \circ \mathsf{spendingPlace}$$

where $\circ$ *represents the DL role constructor of composition. Moreover, any other role term which subsumes those we have pointed out is a valid semantic relation between the concepts which the subsumee role refers to.*

Conditions 1 and 2 of Definition 16 only identify those relations which represent potential candidates for semantically connecting C and D. Stating that C and D are related through R implicitly means that we are deriving and exporting the presence of a concept like $\mathsf{C} \sqcap \exists \mathsf{R}.\mathsf{D}$. Thus, we also require that it is satisfiable with respect to the given ontology.

We now recursively extend the notion of simple semantic relation between concepts by the following definition of *semantic relation*:

Definition 17 (Semantic relation) *Given an ontology* $\mathcal{O} = \langle \text{TBox}, \text{ABox} \rangle$, *two concepts* C *and* D *in* $\mathcal{C}$ *are* semantically related *by a sequence of roles* $\boldsymbol{R} = \langle R_1,\ R_2,\ \ldots,\ R_n \rangle$ *in* $\mathcal{R}^*$ *if there is a concept* E *in* $\mathcal{C}$ *satisfying the following conditions:*

1. R_1 *is a* simple *semantic relation between* C *and* E*;*

2. E *and* D *are semantically related by means of* $\boldsymbol{R}' = \{R_2,\ \ldots,\ R_n\}$.

$\boldsymbol{R}$ *represents a* semantic relation *between concepts* C *and* D.

A particular instance of Definition 17 occurs when $\mathbf{R}$ is the empty sequence, i.e. C is subsumed by D. In this case, we say that C and D are related by an *IsA* relation.

Definition 17 allows to consider semantic relations which are not contemplated by Definition 16. This is the case of one (or more) role of $\mathbf{R} = \{\mathsf{R}_1,\ \mathsf{R}_2,\ \ldots,\ \mathsf{R}_n\}$, other than R_n, representing a *possible* semantic relation according to condition 2 of Definition 16. The presence of *possible* relations in a chain of roles represents one of the reasons which differentiate the concept of *semantic relation* from the *role composition* (i.e. the operator $\circ$). The other divergence concerns the necessity for a semantic relation to specify the "bridge" concepts (e.g. E in Definition 17) univocally, contrary to the semantics of the role composition constructor (see Baader and Sattler [20]).

Summing up, we say that two concepts are *semantically related* if it exists a role R satifying the conditions of Definition 16 or a sequence of roles $\mathbf{R}$ as in Definition 17. In the former case, R is said to be a *simple* semantic relation between the concepts, whereas in the latter case $\mathbf{R}$ respresents a *semantic relation.*

Example 7 *With reference to Example 6, all the semantic relations we have identified are* simple *relations, i.e.* $\boldsymbol{R} = \{\mathsf{R}\}$ *is composed of one element only.* R *is either an atomic role as in the case of* about, $\mathsf{spendingPlace}$ *and* $\mathsf{departure}$ *or a composed role as in the case of* $\mathsf{about} \circ \mathsf{spendingPlace}$. *Furthermore, we can say that* $\mathsf{Insurance}$ *and* Date *are related, too. Indeed, there is a* possible *relation between* $\mathsf{Insurance}$ *and* $\mathsf{Honeymoon}$ *and a* necessary *relation between* $\mathsf{Honeymoon}$ *and* Date. $\boldsymbol{R}$ *is thus composed of two elements,* $\boldsymbol{R} = \langle \mathsf{about}, \mathsf{departure} \rangle$, *and the role of concept* E *in Definition 17 is played by* $\mathsf{Honeymoon}$. *Note that such a semantic relation can only be captured by Definition 17, therefore, it is not a* simple *semantic relation.*

5.3 The algorithm

Once we have formally defined the concept of semantic relation, we introduce the algorithm to extract them from a certain knowledge base. In other words, we show the internal mechanisms of the function $\mathcal{SR}_{\mathcal{O}}$ of Definition 13. Our goal is to identify the semantic relations which occur between two concepts of an ontology $\mathcal{O}$, according to Definition 17. Actually, the $\mathcal{SR}_{\mathcal{O}}$ function requires that only one relation is effectively returned. It should represent the most likely semantic connection between the two input concepts and the relation which best fits in the context. In this version of the algorithm, we consider the relation $\mathbf{R}$ with the minimum number of elements as most probable. In fact, it represents the most intuitive relation, that is the one which requires the smallest reasoning effort for connecting the two concepts. The reasoning minimization is allowed by the minor number of concepts which are needed to be involved in the process (i.e. "bridge" concepts).

Before presenting the algorithm, we must point out some observations related to the decidability of the procedure. Note that:

```
Algorithm 5.1 $\mathcal{SR}$ $(\mathsf{C}, \mathsf{D})$
    ▷ atomic concepts: $\mathsf{C}, \mathsf{D}$
    VarDeclarations
        ontology: $\mathcal{O}$
        relation matrix: $M$
        semantic relation: $\mathbf{R}$
    if ($\mathcal{O} \models \mathsf{C} \sqsubseteq \mathsf{D}$) Return $\langle\rangle$;
    $M \leftarrow$ RELATION–MATRIX($\mathcal{O}$);
    While(true)
        $\mathbf{R} \leftarrow$ NEXT-SHORTEST-PATH($M, \mathbf{R}, \mathsf{C}, \mathsf{D}$);
        if ($\mathbf{R}$=null) Return null;
        if (CONSISTENCY–CHECK($\mathcal{O}, \mathbf{R}, \mathsf{C}, \mathsf{D}$)) Return $\mathbf{R}$;
```

1. in Definition 16, it is stated that any role term R can represent a simple semantic relation between two concepts;
2. condition 2 of Definition 16 requires the existence of a concept E subsuming the target concept D and being a successor of the source concept C;
3. condition 2.b of Definition 16 needs to check the non–existence of a concept F with well–defined properties;
4. definition 17 involves bridge concepts (i.e. concept E) which are qualified as general concept terms.

An algorithm searching semantic relations between concepts must obviously iterate over all the possible values of such generic terms (role term R in case 1, concept term E in case 2 and 4 and concept term F in case 3), thus affecting the decidability of the procedure. In order to guarantee the property, we restrict the analysis to atomic concepts, C in N_C, in cases 2, 3 and 4 and to atomic roles, R in N_R, in case 1. The simplification we make in 1, 2 and 4 does affect the completeness of the algorithm but not its correctness, whereas the trick used in case 3 impacts the correctness, too. In fact, we could find no atomic concepts of the type expressed in condition 2.b but it may exist some complex concept satisfying that condition. This means that the algorithm could return some relations which do not meet point 2.b. In order to maximize the number of relations which are found by the algorithm and to minimize those which do not fulfil condition 2.b, we assume that each complex concept in the ontology is the definition of an atomic concept[1]. Finally, we remark that under the simplifications mentioned, the possible number of elements in a relation $\mathbf{R}$ is limited by the number of concept names in the ontology, i.e. it is finite.

Algorithm 5.1 shows the procedure. The variable $\mathcal{O}$ represents the knowledge source we use for supporting the process of semantic relation discovery. Each

[1] Formally, for each complex concept D in the ontology $\mathcal{O}$, there is a concept name C in N_C and a concept definition $\mathsf{C} \doteq \mathsf{D}$ in the *TBox*.

entailment process of the Algorithms 5.1 and 5.2 can be performed with well-known DL reasoners. In the first step, we check if D subsumes C. It corresponds to the case when $\mathbf{R}$ equals the empty sequence in Definition 17. Step 2 builds the *relation matrix* associated to the ontology. The matrix rows and columns are the concept names of the given ontology, C in N_C, while the cross points contain the role names, R in N_R, which satisfy conditions 1 and 2 of Definition 16, between the corresponding concepts. In other words, the relation matrix contains all the *simple* semantic relations between pairs of concepts. Actually, they are all the possible candidates, since we have not performed the check of consistency yet. Further details will be provided in Algorithm 5.2.

Steps 4–6 compute the "shortest" semantic relation (i.e. $\mathbf{R}$ with the minimum number of elements) between the two input concepts. In particular, step 4 builds a finite oriented graph, $\langle N, E\rangle$, according to the relation matrix. Nodes, n in N, represent concept names, i.e. the relation matrix rows (and columns), and arcs, e in E, the simple semantic relations between the concepts associated to nodes. Note that for each pair of nodes, we can have either no links, if the corresponding concepts do not have any candidate to relate them semantically (i.e. the matrix cross point is empty), or one link, if the corresponding concepts do have some candidates (i.e. the matrix cross point contains one or more elements). We assume that each graph edge has the same cost (weight) and we do not differentiate the possible different relations which connect the same pair of concepts[2]. The NEXT-SHORTEST-PATH function returns the "shortest" relation between the input concepts, if any, while the CONSISTENCY–CHECK procedure checks the consistency of the ontology once added the relation derived. If the relation affects the consistency of the ontology, we search the next "shortest" relation. Thus, the NEXT-SHORTEST-PATH function will have to take care of the previously returned relation in order to identify the next shortest one. Finally, we return *null* if no semantic relation between the input concepts can be found in the given ontology $\mathcal{O}$ (step 5).

In Algorithm 5.2, we analyze the RELATION–MATRIX function in detail. The A_{CN} and the A_{RN} array variables contain all the concept names and all the role names of the input ontology, respectively. Step 1 builds a matrix whose rows and columns represent the elements of A_{CN}. Steps 2–8 fill the matrix. For each row and column, i.e. for each pair of atomic concepts, we search the roles which are candidates for connecting them semantically. Steps 5 and 7 correspond to conditions 1 and 2 of Definition 16. Note that step 7 requires two more iterations over the content of the A_{CN} array in order to check both the presence of the A element and the lack of an element like B. Finally, steps 6 and 8 add those roles which satisfy the conditions (i.e. they are *candidates* for being *simple* semantic relations) to the corresponding matrix cross point.

[2] This is object of studies in the next future.

```
**Algorithm 5.2** RELATION–MATRIX($\mathcal{O}$)
    ▷ *ontology:* $\mathcal{O}$
    **VarDeclarations**
        relation matrix: $M_R$
        array of concept names: $A_{CN}$
        array of role names: $A_{RN}$
    $M_R \leftarrow$ array$[card(A_{CN})][card(A_{CN})]$;
    **for** each row $rw$ in $M_R$ **do**
        **for** each column $cm$ in $M_R$ **do**
            **for** each role R in $A_{RN}$ **do**
                **if** ($\mathcal{O} \models A_{CN}[rw] \sqsubseteq \exists R.A_{CN}[cm]$)
                    $M_R[rw][cm] = M_R[rw][cm] \cup \{R\}$;
                **if** (exists $A \in A_{CN}(\mathcal{O} \models A_{CN}[rw] \sqsubseteq \exists R.A$ and $\mathcal{O} \models A_{CN}[cm] \sqsubset A$
                and not exists $B \in A_{CN}(\mathcal{O} \models A_{CN}[rw] \sqsubseteq \exists R.B$ and $\mathcal{O} \models B \sqsubset$
                $A)))$
                    $M_R[rw][cm] = M_R[rw][cm] \cup \{R\}$;
    **Return** $M_R$;
```

5.4 Some theoretical remarks

We believe that ontology reasoning in the field of semantic explicitation must be performed in *open world assumption* because we cannot assume that practical ontologies contain all the possible knowledge about the world. Thus, we must consider the possibility that something which is not asserted in the knowledge base is true anyhow. This necessary assumption makes a concept as $C \sqcap \exists R.D$ satisfiable most of the times, for any C, D and R in the ontology. In fact, in order to make the concept unsatisfiable, it should be explicitly stated, in some way, that $C \sqsubseteq \forall R.\neg D$ (assumed that C and D are satisfiable). However, we cannot allow that each role is considered as a candidate for semantically relating any pair of concepts (thus making it an actual semantic relation most of the times, given the above consideration). This would make the work of relation extraction dummy because we would not employ the knowledge contained in the ontology effectively. Instead, our goal is to extract relations with respect to which the ontology tells something which is not tautological. This is the difference between relevant and non–relevant relations as far as our methodology considers them. Relevant relations are the previously named *necessary* and *possible*, while the non–relevant are called *tautological.* This section presents the difference among these three types of relations from an ontological point of view.

Necessary relations are those which can be inferred from the ontology as in the following examples:

- $TBox_1 = \{E \sqsubseteq \exists R.D,\ C \sqsubseteq E\} \models C \sqsubseteq \exists R.D$;
- $TBox_2 = \{C \sqsubseteq \exists R.E,\ E \sqsubseteq D\} \models C \sqsubseteq \exists R.D$.

The evidence of the reasons supporting the inference can be achieved by considering number restrictions in Description Logic. Consider the following two knowledge bases in a DL with qualified number restrictions. They represent the extension of the previous examples with cardinalities ($n > 0$, m is not constrained):

- $\mathit{TBox}'_1 = \{\mathsf{E} \sqsubseteq (\geq n\mathsf{R}.\mathsf{D} \sqcap \leq m\mathsf{R}.\mathsf{D}),\ \mathsf{C} \sqsubseteq \mathsf{E}\}$;
- $\mathit{TBox}'_2 = \{\mathsf{C} \sqsubseteq (\geq n\mathsf{R}.\mathsf{E} \sqcap \leq m\mathsf{R}.\mathsf{E}),\ \mathsf{E} \sqsubseteq \mathsf{D}\}$.

Inferential reasoning leads to the following conclusions:

- $\mathit{TBox}'_1 \models \mathsf{C} \sqsubseteq (\geq n\mathsf{R}.\mathsf{D} \sqcap \leq m\mathsf{R}.\mathsf{D})$;
- $\mathit{TBox}'_2 \models \mathsf{C} \sqsubseteq (\geq n\mathsf{R}.\mathsf{D})$.

Both the minimum and the maximum cardinalities are inherited by the inferred relation in the former case. In the latter case, nothing can be deduced about the maximum cardinality. The inheritance of the minimum cardinality represents the common feature of the two examples and the implicit reason which validates the inference in the case of lack of number restrictions.

We consider this type of relation between C and D as *necessary* and it is considered in the first condition of Definition 16. It defines R as a simple semantic relation between C and D, if all the instances of C must be R–antecedents with respect to an instance of D, for any model of the given ontology. In other words, it captures all those relations which are required to occur for each instance of the source concept C and whose range is represented by an element of the target concept D. These relations can be directly inferred from the knowledge base (*necessary* relations).

Now consider two more examples about inferential reasoning (similarly to what introduced in Section 5.2, we use $\mathsf{C} \sqsubset \mathsf{D}$ to denote that $\mathsf{C} \sqsubseteq \mathsf{D}$ and $\mathsf{C} \sqcap \mathsf{D} \sqsubseteq \bot$):

- $\mathit{TBox}_3 = \{\mathsf{C} \sqsubseteq \exists\mathsf{R}.\mathsf{E},\ \mathsf{D} \sqsubset \mathsf{E}\} \models \mathsf{C}\ ?\ \mathsf{D}$;
- $\mathit{TBox}_4 = \{\mathsf{E} \sqsubseteq \exists\mathsf{R}.\mathsf{D},\ \mathsf{E} \sqsubset \mathsf{C}\} \models \mathsf{C}\ ?\ \mathsf{D}$.

In both cases, we cannot deduce any kind of relation between C and D. Now, consider their extension with qualified number restrictions ($n > 0$, m is not constrained):

- $\mathit{TBox}'_3 = \{\mathsf{C} \sqsubseteq (\geq n\mathsf{R}.\mathsf{E} \sqcap \leq m\mathsf{R}.\mathsf{E}),\ \mathsf{D} \sqsubset \mathsf{E}\}$;
- $\mathit{TBox}'_4 = \{\mathsf{E} \sqsubseteq (\geq n\mathsf{R}.\mathsf{D} \sqcap \leq m\mathsf{R}.\mathsf{D}),\ \mathsf{E} \sqsubset \mathsf{C}\}$;

Inferential reasoning leads to the following conclusions:

- $\mathit{TBox}'_3 \models \mathsf{C} \sqsubseteq (\geq 0\mathsf{R}.\mathsf{D} \sqcap \leq m\mathsf{R}.\mathsf{D})$;
- $\mathit{TBox}'_4 \models \mathsf{C} \sqsubseteq (\geq 0\mathsf{R}.\mathsf{D})$;

The inference about the relation between C and D in the latter case is *tautological*, i.e. the minimum cardinality is 0 and the maximum cardinality is unlimited. This means that in any model of the ontology, an instance of C could have no R-subsequents of the type of D or an unlimited number, that says nothing (we are reasoning in open world assumption). On the contrary, the relation inherits the maximum cardinality in the former case. The minimum cardinality set to 0 prevents from inferring the presence of the relation if cardinalities are not considered (i.e. the relation is not *necessary*), however what has been deduced is not tautological because of the inheritance of the maximum cardinality.

We do not consider *tautological* relations because it is as if the ontology did not say anything (i.e. we would find such relations for any pair of concepts unless explicitly forbidden as explained at the beginning of the section), while we identify a *possible* relation in those cases where the maximum cardinality is preserved (although the minimum is set to 0). The second condition of Definition 16 approximates these cases.

More complex situations must be dealt with when considering different subsumees of the same concept, one of which representing the range of an existential constraint. Consider, for example, the following knowledge base:

$$\mathit{TBox}_5 = \{\mathsf{C} \sqsubseteq \exists \mathsf{R.E},\ \mathsf{E} \sqsubset \mathsf{F},\ \mathsf{D} \sqsubset \mathsf{F},\ \mathsf{E} \sqcap \mathsf{D} \sqsubseteq \bot\}$$

According to what we have stated about *possible* relations, we would be tempted to consider concepts C and D semantically related by means of R. In fact, C is inferentially related to F and D is a subset of F. Thus, we may conclude that a *possible* relation between C and D exists. Unfortunately, a more careful analysis denies this conclusion. Intuitively, if R is constrained by minimum and maximum cardinalities, n and m respectively, the inferred relation between C and F loses the superior boundary (see the case of TBox_2) and the relation between C and D through F does not result *necessary* (i.e. minimum cardinality set to 0). Thus, the final relation is *tautological*. Condition 2.b of Definition 16 avoids these misunderstandings by requiring that E is the most specific concept representing the range of the existential constraint for C with respect to the role R (i.e. there is *no* concept F such that: $\mathcal{O} \models \mathsf{C} \sqsubseteq \exists \mathsf{R.F}$ and $\mathcal{O} \models \mathsf{F} \sqsubset \mathsf{E}$).

Chapter 6

CtxMatch 2

6.1 Algorithm and data structures

In Chapter 4, we have discussed the semantic method we apply for matching hierarchical classifications. The methodology is based on the derivation of the node meaning and on their logical comparison. Since semantic comparison is simply a matter of reasoning on Description Logic terms, the hardest step is represented by semantic explicitation. In Section 4.1.4, we have introduced the macro–steps which are required to perform the process. We have identified the necessity of (i) analyzing the grammatical structure of natural language strings, (ii) associating each node with the respective lexicon, (iii) enriching the schema with the semantic relations which connect the different concepts and (iv) finally filtering and selecting the most appropriate interpretation for the nodes of the structure. In this chapter, we discuss the way of effectively executing the single macro–steps. In some cases, our contribution is only in terms of procedures (e.g. building of the parse tree, semantic enrichment), while in other cases, we propose the heuristic criteria we adopt for simulating those steps which humans usually perform according to their own sensibility and experience (e.g. semantic filtering, choosing one interpretation).

Before presenting the algorithm details, let us introduce the notion of semantic ACH formally. We have already discussed that the process of meaning derivation requires a sequence of steps aimed at collecting all the possible semantic information. A *semantic ACH* is simply a data structure whose purpose is to store the information collected. The algorithms we describe in this chapter elaborate the content of the semantic ACH and generate the formalization of the node interpretations.

We refer to the elements of a node $n = \langle Word, Lemma, SynCat, POS, Senses\rangle$ of a parse tree $P_{\mathcal{L}}$ as $Word(n)$, $Lemma(n)$, $SynCat(n)$, $POS(n)$ and $Senses(n)$, respectively. Moreover, 'word$_{Lemma}$' and 'word$_{POS}$' represent the lemma and the part of speech of the word 'word', respectively.

Definition 18 (Semantic ACH) *The* semantic labeling of an ACH $H = \langle K, E, \text{att}, \text{lab}\rangle$ *is the 5-tuple* $H_s = \langle H, PT, E_{ACH}, E_{PT}, \text{lab}_s\rangle$*, where:*

1. $PT : K \rightarrow \{P_{\mathcal{L}}\}$*, is a function which relates each node* $k \in K$ *to a parse tree,* $P_{\mathcal{L}}$*;*

2. E_{PT} *is a set of 3-tuples* $\langle\langle k, n_A, s_A\rangle, \langle k, n_B, s_B\rangle, \boldsymbol{R}\rangle$*, where* $k \in K$*,* n_A *and* n_B *are different nodes of* $PT(k)$*,* $s_A \in \text{Senses}(n_A)$*,* $s_B \in \text{Senses}(n_B)$ *and* $\boldsymbol{R} = \mathcal{SR}_{\mathcal{O}}(s_A, s_B)$*;*

3. E_{ACH} *is a set of 3-tuples* $\langle\langle k_A, n_A, s_A\rangle, \langle k_B, n_B, s_B\rangle, \boldsymbol{R}\rangle$*, where* k_A *and* k_B *are different nodes of* H*,* n_A *and* n_B *are nodes of* $PT(k_A)$ *and* $PT(k_B)$*, respectively,* $s_A \in \text{Senses}(n_A)$*,* $s_B \in \text{Senses}(n_B)$ *and* $\boldsymbol{R} = \mathcal{SR}_{\mathcal{O}}(s_A, s_B)$*;*

4. $\text{lab}_s : K \rightarrow \mathcal{C}$*, is a function that relates each node* $k \in K$ *to a concept of the ontology* $\mathcal{O}$*.*

The resulting structure, H_s*, is called* semantic ACH*.*

PT associates each node of an ACH to a parse tree representing the *grammatical* structure of the node and storing the *lexicon* of its elements with respect to a *lexicon* function, $\mathcal{L}$. E_{PT} is a set of *ontological* relations connecting the different elements of the same node. Similarly, E_{ACH} is a set of relations connecting the different nodes of the ACH. Finally, lab_s represents the result of the semantic explicitation process, i.e. the interpretation we have assigned to each node k of the structure, i.e. $\mathcal{I}(k)$.

Next sections describe the procedures which add semantic information to the ACH and elaborate its content. Section 6.2 describes the mechanisms to define the PT function, Section 6.3 those to enrich the ACH with semantic relations (i.e. defines E_{PT} and E_{ACH}) and Section 6.4 presents the heuristic criteria and the algorithm to filter those concepts which do not fit in the context. Section 6.5 further discards concepts and relations in order to select a unequivocal interpretation. Finally, Section 6.6 builds the concept term describing the content of the ACH nodes by defining the lab_s function.

6.2 Parse tree building

The first step in semantic explicitation is to analyze labels and attributes in order to identify the corresponding constituents (e.g. single– and multi–words) and the grammatical dependencies existing between such elements. This is a common problem in the field of NLP as already claimed in Section 4.1.1. The parsing of natural language strings leads to the building of a *parse tree* (see Definition 15) representing the grammatical structure of the phrase, identifying the grammatical nature of its elements and associating each of them with the corresponding lexicon.

In this section, we describe how parse trees can be composed, i.e. we present the function PARSE–TREE of Algorithm 4.1. In particular, we rely on existing functions in the field of NLP both to perform the grammatical analysis of the labels (and attributes) and to add lexical information to the elements of a parse tree. We refer to such functions as NLP–PARSING. Note that we also assume that a NLP–PARSING function is able to recognize and deal with acronyms, abbreviations, compact notations and all those expressions which are commonly used in labels. Our contribution is to include the node attributes as part of a parse tree, since they can be considered as modifiers of the concept expressed by the node label. Actually, attribute fillers (i.e. F) are modifiers of the node subject(s), whereas attribute names (i.e. id) represent the semantic relation connecting the node concept to the attribute filler. Thus, we append attribute fillers to the parse tree structure as children of the label subject(s) (i.e. parse tree elements whose synctactical category is set to *Head*) and we add attribute names to the set of semantic relations (i.e. E_{PT}). Unfortunately, attribute names are natural language strings, therefore we do not know their actual meaning in the current context. In other words, we would need a process of word sense disambiguation for them, too. In this version of the algorithm, we simply refer to them as the disjunction of all their possible meanings.

We now analyze in detail the function PARSE–TREE of Algorithm 6.1. It takes an ACH, $H = \langle K, E, att, lab \rangle$, and returns a semantic ACH, $H_s = \langle H, PT, E_{ACH}, E_{PT}, lab_s \rangle$, where E_{ACH}, E_{PT} and lab_s are undefined. Its goal is to build the PT function, i.e. to associate each node of H to a parse tree.

The same set of operations (steps 2–19) is repeated independently for each node of the input ACH structure. Step 2 parses the label, $lab(k)$, of a node k and builds a *parse tree* object. Note that the root of a parse tree can be also a virtual node @. The symbol @ is used when the phrase contains more than one subject. In that case, all the subjects are identified as children of the virtual node. Actually, we will also use the node @ to introduce sub–trees representing attributes. The cause will be clearer in Section 6.6, where we will present the symbol semantics. Steps 3–19 aim at including into the parse tree of a node its attributes $a = \langle id, F \rangle$. Step 5 sets the variable $\mathbf{R}$ (i.e. the semantic relation between the label subject and the attribute filler) to the disjunction of all the possible meanings of the attribute name, id. We assume that the attribute name is a string which can be directly mapped onto ontological concepts, i.e. it is a lexicon entry. This guarantees that no further parsing is required for the id string, except for the recognition of its lemma and its part of speech, id_{Lemma} and id_{POS}, respectively. Step 6 creates the virtual node, @, which is used to precede any attribute and which will be finally appended to the parse tree headers in step 16. Steps 7–15 build the tree structure representing the attribute a. It will be then appended to the virtual node @ (see steps 9 and 14). Steps 8–10 and 12–15 represent the different algorithm behaviour according to the possible values of the attribute filler F, namely $\emptyset$, F_f and F_r. In case $F = \emptyset$, we assume it represents the ontological top concept $\top$ (i.e. the attribute value can be of any type). Thus, we create a node corresponding to the *top* concept (step 8) and we semantically link it to the virtual node (step 10) by means of the relation $\mathbf{R}$. Note that the syntactical category (*SynCat*) of the attributes must be always set to *Mod* because they always represent modifiers

Algorithm 6.1 PARSE–TREE (H)

```
    ▷ $ACH$: $H = \langle K, E, \mathrm{att}, \mathrm{lab} \rangle$
    VarDeclarations
        semantic $ACH$: $H_s = \langle H, PT, E_{ACH}, E_{PT}, lab_s \rangle$
        parse tree: $P_{\mathcal{L}} = \langle N, G \rangle$
        semantic relation: $\mathbf{R}$
        parse tree node: $v$
for each node $k \in K$ do
    $P_{\mathcal{L}} \leftarrow$ NLP–PARSING$(lab(k))$;
    for each node $n \in N$ where $SynCat(n) = Head$ do
        for each attribute $a = \langle id, F \rangle \in att(k)$ do
            $\mathbf{R} \leftarrow \langle \bigsqcup_{\mathsf{C} \in \mathcal{L}_{\mathcal{O}}(id_{Lemma}, id_{POS})} \mathsf{C}\} \rangle$;
            $m \leftarrow @$;
            if $(F = \emptyset)$ then
                $v \leftarrow \langle \_, \_, Mod, \_, \{\top\} \rangle$;
                $P_{\mathcal{L}} \leftarrow$ APPEND$(m,v)$;
                $E_{PT} = E_{PT} \cup \{\langle @, \langle k, v, \top \rangle, \mathbf{R} \rangle\}$;
            else
                for each $f \in F_f$ (or $f = F_r$) do
                    $v \leftarrow$ NLP–PARSING$(f)$;
                    $P_{\mathcal{L}} \leftarrow$ APPEND$(m,v)$;
                    for each $s_v \in Senses(v)$ do $E_{PT} = E_{PT} \cup$
                    $\{\langle @, \langle k, v, s_v \rangle, \mathbf{R} \rangle\}$;
            $P_{\mathcal{L}} \leftarrow$ APPEND$(n,m)$;
            for each $s_n \in Senses(n)$ do
                $E_{PT} = E_{PT} \cup \{\langle \langle k, n, s_n \rangle, @, \mathbf{R} \rangle\}$;
    $PT(k) \leftarrow P_{\mathcal{L}}$;
Return $H_s$;
```

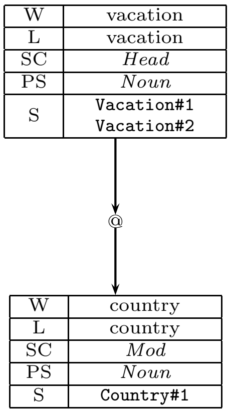


Figure 6.1: Parse tree of `Vacations`

of the label subject(s). The cases $F = F_f$ and $F = F_r$ are managed in the same way, since F_f is a set of strings and F_r, which is a string, can be considered a single–element set. In step 13, we create a node for each element f of this set. The node is the result of the parsing procedure applied to the element. Note that f could be a string of any complexity (e.g. 'Republican countries and nations'), thus the result of the parsing of a filler is a parse tree rooted at v. For the sake of simplicity, we assume that it is a single–node structure, however, the extension to the general case is quite immediate. In step 15, we enrich the set of the parse tree relations, E_{PT}, with the links between the virtual node @ and all the possible senses of f. Note that if $F = F_r$, the procedure results in a single iteration of the cycle at step 12. Once we have created the tree rooted at @ representing the attribute a, we append it to the existing parse tree structure (step 16). In step 18, we add the relation $\mathbf{R}$ from all the senses of the subject n of the node label to the virtual node (preceding the attribute). Note that at this step, we do not know not only the meaning which an attribute name refers to, but also the concept which is represented by the subject of the ACH node and by the attribute filler. Hence, we link any sense of the subject(s) to any sense of the filler element(s) by means of the node @. Finally, step 20 returns the semantic ACH with the PT function defined.

Example 8 *Consider the node* `Vacations`*, k, of the ACH in Figure 2.1. The resulting parse tree is represented in Figure 6.1. Two elements have been identified: the node which refers to the label 'vacations' and the node which refers to the filler 'country'. 'Vacation' is both the word and the lemma, it is the header of the label and a noun. Moreover, we relate it to the two concepts* `Vacation#1` *and* `Vacation#2`*. The ACH node has one attribute, $a = \langle location, Country\rangle$. We link the corresponding node to the parse tree header by means of the virtual node @ and we set its syntactical category to* Mod*. Finally, given the lexicon,* `location#1`*, of the attribute name 'location', we enrich the E_{PT} set with three*

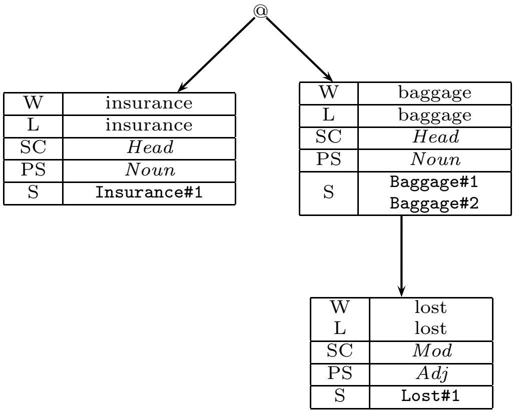


Figure 6.2: Parse tree of `Insurance and lost baggage`

triples representing the semantic relation from the node label to the attribute filler, namely, $\langle\langle k, n,$ `Vacation#1`$\rangle, @,$ `location#1`$\rangle$*,* $\langle\langle k, n,$ `Vacation#2`$\rangle, @,$ `location#1`$\rangle$*,* $\langle @, \langle k, m,$ `Country#1`$\rangle,$ `location#1`$\rangle$*, where* n *is the parse tree element which refers to 'vacation' and* m *to 'country'.*

Example 9 *In this example, we consider the node* `Insurance and lost baggage` *of Figure 2.1. The resulting parse tree is depicted in Figure 6.2. It has no attributes but two headers and a modifier. The headers are respectively 'insurance' and 'baggage' which are rooted at @. The adjective 'lost' generates one more parse tree node: it is a modifier and we can identify the word it refers to by means of the tree structure: 'baggage'.*

6.3 Semantic enrichment

The research of the semantic relations between the elements of an ACH is the major enhancement that our methodology introduces with respect to traditional techniques of matching. Indeed, we propose both to disambiguate and to build the interpretation of a node in the structure by finding and using the semantic relations between those concepts which are contextually involved in the node meaning. In particular, we search semantic relations (i) between the elements belonging to the grammatical structure of an ACH node (i.e. *local semantic enrichment*) and (ii) between the concepts which are associated to the ACH nodes (i.e. *global semantic enrichment*). With respect to local enrichment, we intuitively need to discover relations between words in labels and attributes in order to make the meaning of natural language strings explicit. Consider, for example, the following part of the label of the node `Insurance and lost baggage`: 'lost baggage'. We know that 'lost' is an adjective for 'baggage' but we are also interested in realizing how the adjective specifies the corresponding noun. In this case, we could deduce that 'lost' represents the *state* of the baggage, as

```
Algorithm 6.2 LOCAL–SEMANTIC-ENRICHMENT($H_s, k$)
    ▷ semantic ACH: $H_s = \langle H, PT, E_{ACH}, E_{PT}, \mathrm{lab}_s \rangle$
    ▷ Node: $k \in H$
    VarDeclarations
        parse tree: $P_{\mathcal{L}} = \langle N, G \rangle$
        semantic relation: $\mathbf{R}$
    $P_{\mathcal{L}} \leftarrow PT(k)$
    for each pair $m, n \in G$ where $n =$CHILD $(m)$
        for each pair $s_m \in Senses(m), s_n \in Senses(n)$
            $\mathbf{R} \leftarrow \mathcal{SR}_{\mathcal{O}}(s_m, s_n)$;
            if ($\mathbf{R}$ != null)
                $E_{PT} \leftarrow E_{PT} \cup \{\langle \langle k, m, s_m \rangle, \langle k, n, s_n \rangle, \mathbf{R} \rangle\}$;
    Return $H_s$
```

```
Algorithm 6.3 GLOBAL–SEMANTIC-ENRICHMENT($H_s$)
    ▷ semantic ACH: $H_s = \langle H, PT, E_{ACH}, E_{PT}, \mathrm{lab}_s \rangle$
    VarDeclarations
        semantic relation: $\mathbf{R}$
    for each pair $k, k' \in E$
        for each pair $m \in PT(k)$, $n \in PT(k')$ ($SynCat(m) = SynCat(n) =$
        $Head$)
            for each pair $s_m \in Senses(m), s_n \in Senses(n)$
                $\mathbf{R} \leftarrow \mathcal{SR}_{\mathcal{O}}(s_m, s_n)$;
                if ($\mathbf{R}$ != null)
                    $E_{ACH} \leftarrow E_{ACH} \cup \{\langle \langle k, m, s_m \rangle, \langle k', n, s_n \rangle, \mathbf{R} \rangle\}$;
    Return $H_s$
```

well as we could say that 'large' represents its *size*, if we had a label like 'large baggage'. Moreover (considering the global enrichment), the meaning of a node in an ACH depends on its context, thus its semantic relation with the corresponding ancestors is fundamental for its interpretation as pointed out since the first lines of our dissertation.

Semantic enrichment is supported by the $\mathcal{SR}_{\mathcal{O}}$ function which returns the most probable semantic relation between two concepts with respect to an ontology $\mathcal{O}$.

LOCAL–SEMANTIC–ENRICHMENT in Algorithm 6.2 takes a semantic ACH, $H_s = \langle H, PT, E_{ACH}, E_{PT}, lab_s \rangle$, and a node $k \in H$, as inputs whereas GLOBAL–SEMANTIC–ENRICHMENT in Algorithm 6.3 takes the semantic ACH structure only. The former function fills the E_{PT} set with the relations between the elements of the node k, while the latter searches the inter–node relations, i.e. the elements of the E_{ACH} set. The two procedures are actually very similar.

In both cases, the structures on which semantic enrichment works are trees: the parse tree of a node in local semantic enrichment and the input ACH in global semantic enrichment. Step 2 in Algorithm 6.2 corresponds to step 1 in Algorithm 6.3. Indeed, step 2 triggers a cyclic repetition applied to each node pair of the parse tree, while step 1 does the same for each node pair of the ACH. In this version of the algorithm, we only search relations between nodes on the same path[1]: of the parse tree in one case, i.e. $m, n \in G$, and of the ACH in the other one, i.e. $k, k' \in E$. The main difference between the two algorithms can be identified in the code `where n=CHILD(m)` of step 2 of Algorithm 6.2. In parse trees, the structure suggests the entities between which the relation must be searched. Indeed, the edges of the structure start from a subject and point to its modifier(s) and analogously, semantic relations are directional: from the relation subject to the object. Therefore, we can seek only the relations from a node to its children which are all and only the modifiers of that node (given the way parse trees are built). The same does not hold for global enrichment where we must also search relations from a node to its ancestors and between nodes at any distance. Step 2 of Algorithm 6.3 selects the header nodes of the parse trees of the ACH nodes we are analysing. In fact, we are only interested in relating the subjects of different nodes since modifiers can only refer to an element of the corresponding ACH node. Steps 3–6 of both algorithms extract the relations between each pair of senses of the nodes selected and, if any, they add it to the corresponding set: E_{PT} in local enrichment and E_{ACH} in global enrichment. The nodes selected are two different nodes of the same parse tree in the former case and two headers of two different nodes of the ACH in the latter. Note that if one of the two nodes is a virtual node @, the algorithm does not perform the usual process: if @ represents the parse tree root, there is no point in searching relations from @ to its children, while if @ introduces an attribute we do not have to link it to any other concept since the semantic relation from the label subject(s) to the attributes have already been identified and considered in the function PARSE–TREE. Finally, step 7 in both algorithms returns the semantic ACH where either the E_{PT} set or the E_{ACH} set have been defined.

Example 10 *Consider the node* `Insurance and lost baggage` *of the ACH in Figure 2.1 and its context* `Vacations`*. The tree structure in Figure 6.3 is a partial representation of the corresponding semantic ACH.*

The research of elements of E_{PT} concerning the node `Vacations` *does not give any result because the corresponding parse tree has only one node which does not represent an attribute: the concept* `Country#1` *is an attribute filler, thus its relation with the header is already identified by the attribute name. Instead, the relations in* `Insurance and lost baggage` *are not explicit. Suppose that* local semantic enrichment *finds the relation $\boldsymbol{R} = \{$*`state#3`*$\}$ between the concept* `Baggage#1` *and the concept* `Lost#1`*. Note that the relation is found between a node (i.e. the one which refers to the concept* `Baggage#1`*) and its*

[1] Future work will investigate the possibility of enhancing the algorithm performances by considering relations between non–directly dependent nodes (i.e. nodes which do not belong to the same path). In the case of ACH trees this corresponds to the extension of the concept of *context*.

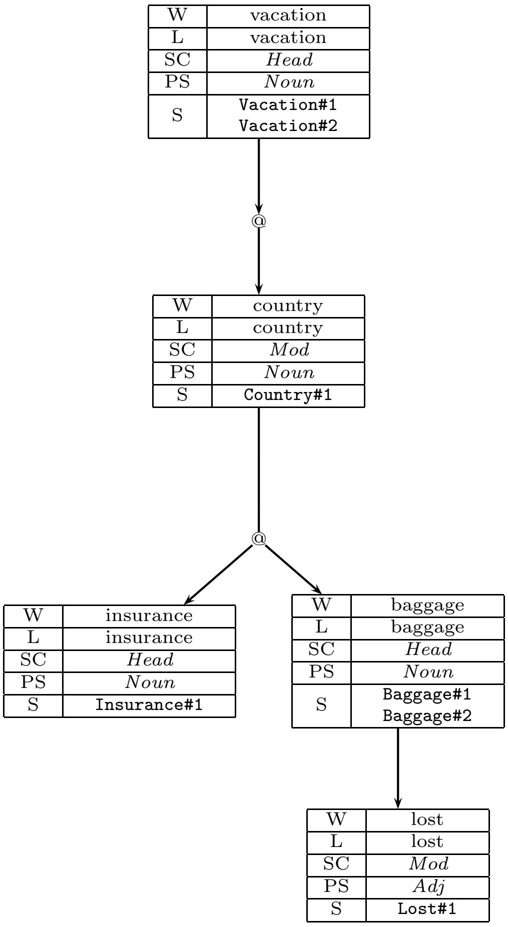


Figure 6.3: Part of the semantic ACH associated to the context of `Insurance and lost baggage`

child (i.e. the one which refers to the concept `Lost#1`*).* Global semantic enrichment *searches relations between the headers of the two parse trees, i.e. between the senses of 'vacation' and those of 'insurance' and 'baggage'. We suppose that $\mathcal{SR}_{\mathcal{O}}$ returns a non-null element for the concepts* `Insurance#1` *and* `Vacation#1`*,* ***R'*** $=$ {`about#1`}*, and between* `Vacation#1` *and* `Insurance#1`*,* ***R"*** $=$ {`organizedBy#1`}*. R' and R'' are then added to the E_{ACH} set.*

6.4 Semantic filtering

Semantic filtering is the opposite process compared to semantic enrichment. Summing up: (i) PARSE–TREE has associated each element (i.e. *lexicon* entry) to all the corresponding concepts, (ii) SEMANTIC–ENRICHMENT has detected all the possible relations between them, (iii) semantic filtering aims at filtering both of them. The presence of many concepts for the same word and the presence of many possible semantic relations are a source of ambiguity to determine the meaning of the nodes. At this step, the semantic ACH contains all the information we need to make the semantics of the nodes explicit but unfortunately we can interpret them in many different ways. The goal of semantic filtering is to disambiguate the sense of words similarly to how human beings do by examining the context.

The criteria for filtering the possible meanings are heuristic and their choice and definition strongly influence the algorithm performances. Let us suppose we have collected all the information which is necessary to establish the meaning of the structure (it is a matter of linguistic resources and ontology selection in the functions PARSE–TREE and SEMANTIC-ENRICHMENT), we may achieve the maximum recall by providing all the possible interpretations for each node but we would lack precision. Therefore, we must reduce the ambiguity in order to balance the two performance indexes. The heuristic criteria we adopt are based on the intuitive assumption that the most likely meaning of a word is the one which best fits in its own context.

We apply two different methodologies which are respectively based on the two following intuitions:

Semantic rule: suppose we have a natural language word, w, which is associated to more than one concept (i.e. possible meanings). We assume that the right meaning of w is one of those which are semantically related to some other elements of the context. In other words, we trust the phase of semantic enrichment and we only keep the word interpretations which are linked to the rest of the context.

Subsumption rule: suppose we have a natural language word, w, which is associated to more than one concept and suppose that one of them, C in the lexicon of w, is related by an *IsA* or *PartOf* relation with another element of the context. Hence, we assume that C is the right meaning of w. In other words, we are biased towards structures which are based on *IsA* and *PartOf* relations.

```
Algorithm 6.4 LOCAL–SEMANTIC–FILTERING($H_s, k$)
    ▷ semantic ACH: $H_s = \langle H, PT, E_{ACH}, E_{PT}, \mathrm{lab}_s \rangle$
    ▷ node: $k \in H$
    VarDeclarations
        parse tree $P_{\mathcal{L}} = \langle N, G \rangle$
    $P_{\mathcal{L}} \leftarrow PT(k)$
    for each $n \in N$
        for each $s \in \mathit{Senses}(n)$
            if $(\neg\exists e\ (e = e^{PT}_{\langle k,n,s\rangle\leftrightarrow}) \sqcap \exists s'\ \exists e\ (e = e^{PT}_{\langle k,n,s'\rangle\leftrightarrow}))$
            remove $s$ from $\mathit{Senses}(n)$ and each $e = e_{\langle k,n,s\rangle\leftrightarrow}$;
            if $(\forall e\ (e = e^{PT}_{\langle k,n,s\rangle\leftrightarrow} \Rightarrow (\mathbf{R}_e \neq \mathit{IsA}\ \sqcap\ \mathbf{R}_e \neq \mathit{PartOf})) \sqcap$
              $\exists s'\exists e\ (e = e^{PT}_{\langle k,n,s'\rangle\leftrightarrow} \sqcap (\mathbf{R}_e = \mathit{IsA} \sqcup \mathbf{R}_e = \mathit{PartOf})))$
            remove $s$ from $\mathit{Senses}(n)$ and each $e = e_{\langle k,n,s\rangle\leftrightarrow}$;
    Return $H_s$
```

Note that such criteria do not fully solve the ambiguity because each of them continues considering all the concepts which satisfy its conditions and does not discard any meaning if no element satisfies them.

We distinguish between two phases of semantic filtering: *local filtering* and *global filtering*. The former considers the label and the attributes of a node independently from the rest of the ACH, whereas the latter takes care of all (or part of) the structure. Let w be a word of the label or of an attribute of a node k. Local filtering considers the other elements of the label and of the attributes of k as context of the disambiguation process of w. Instead, such a context is represented by the elements of the other nodes of the ACH in global filtering. Actually, we only consider the nodes in the context of k and the corresponding headers. Anyway, local and global filtering are based on the same above criteria and work very similarly.

LOCAL–SEMANTIC–FILTERING in Algorithm 6.4 takes a semantic ACH, $H_s = \langle H, PT, E_{ACH}, E_{PT}, lab_s \rangle$ and a node $k \in H$, whereas GLOBAL–SEMANTIC–FILTERING in Algorithm 6.5 the ACH structure only. The former filters the senses of the elements in $PT(k)$ and the corresponding relations in E_{PT}, while the latter the senses of the headers of the nodes in H and the relations in E_{ACH}.

We have introduced a compact notation for the relations in E_{ACH} and in E_{PT} to simplify the reading. $e = e^{ACH}$ indicates that e belongs to the E_{ACH} set, while $e = e^{PT}$ indicates that e belongs to the E_{PT} set. If the top index is not present, e can be an element both of E_{ACH} and of E_{PT}. $e = e_{\langle k,n,s\rangle\rightarrow}$ means that e has $\langle k, n, s\rangle$ as first element of its triple (i.e. the antecedent of this relation), while $e = e_{\langle k,n,s\rangle\leftarrow}$ means that e has $\langle k, n, s\rangle$ as second element of its triple (i.e. the subsequent of this relation). $e = e_{\langle k,n,s\rangle\leftrightarrow}$ is used if $\langle k, n, s\rangle$ can be both the antecedent and the successor of e. Note that if the top index or an element of the triple in the bottom index are not present, it means that their value can be any. Finally, $\mathbf{R}_e$ indicates the third element of a triple in E_{PT} or E_{ACH}, i.e. the semantic relation.

```
Algorithm 6.5 GLOBAL–SEMANTIC–FILTERING($H_s$)
    ▷ semantic ACH: $H_s = \langle H, PT, E_{ACH}, E_{PT}, \text{labs}_s \rangle$
1   for each $k \in H$
2       for each $n \in PT(k)$ where $SynCat(n) = Head$
3           for each $s \in Senses(n)$
4               if $(\neg\exists e\ (e = e_{\langle k,n,s\rangle\leftrightarrow}) \sqcap \exists s'\ \exists e\ (e = e^{ACH}_{\langle k,n,s'\rangle\leftrightarrow}))$
5               remove $s$ from $Senses(n)$ and each $e = e_{\langle k,n,s\rangle\leftrightarrow}$;
6               if $(\forall e\ (e = e_{\langle k,n,s\rangle\leftrightarrow} \Rightarrow (\mathbf{R}_e \neq IsA\ \sqcap\ \mathbf{R}_e \neq PartOf)) \sqcap$
7                  $\exists s'\exists e\ (e = e^{ACH}_{\langle k,n,s'\rangle\leftrightarrow} \sqcap (\mathbf{R}_e = IsA \sqcup \mathbf{R}_e = PartOf)))$
8               remove $s$ from $Senses(n)$ and each $e = e_{\langle k,n,s\rangle\leftrightarrow}$;
9   Return $H_s$
```

E_{PT}	E_{ACH}
⟨Vacation#1, @, ⟨location#1⟩⟩	⟨Insurance#1, Vacation#1, ⟨about#1⟩⟩
⟨Vacation#2, @, ⟨location#1⟩⟩	⟨Vacation#1, Insurance#1, ⟨organizedBy#1⟩⟩
⟨@, Country#1, ⟨location#1⟩⟩	
⟨Baggage#1, Lost#1, ⟨state#3⟩⟩	

Table 6.1: E_{PT} and E_{ACH} sets

The algorithms apply the same rules but with respect to different sets of relations: E_{PT} in case of local filtering and E_{ACH} in case of global filtering. Algorithm 6.4 focuses on each sense of each node of a given parse tree (steps 1–3), while Algorithm 6.5 on each sense of each header (steps 1–3). Steps 4–5 and 6–8 of both the algorithms are respectively the *semantic rule* and the *subsumption rule*. Note that SEMANTIC–FILTERING removes a sense s when it matches the rule conditions both locally and globally in *global filtering* (steps 4 and 6–7: $e = e_{\langle k,n,s\rangle\leftrightarrow}$), while it only checks local relations in *local filtering* (steps 4 and 6–7: $e = e^{PT}_{\langle k,n,s\rangle\leftrightarrow}$). When it removes a sense s from the corresponding set, it also removes all the relations involving it both as antecedent and as subsequent. Finally, step 9 returns the input semantic ACH with a subset of the pre–existing senses and relations.

Example 11 *Consider the results of the semantic enrichment process of Example 10. Figure 6.4 represents part of the current semantic ACH. In particular, we have highlighted the concepts associated to each parse tree node.*

We can summarize the composition of the sets E_{PT} and E_{ACH} as in Table 6.1 (we omit the ACH nodes and the parse tree nodes in the triples).

By applying the heuristic rules of filtering for each parse tree (i.e. locally), we can notice that both the concepts of 'vacation' are internally related with Country#1 *through the node @, while only the concept* Baggage#1 *of 'baggage' is related to other elements of the corresponding parse tree. Thus, we only trigger the* semantic rule *in the latter case and we remove the concept* Baggage#2*. No global rule is triggered. The filtering result is shown in Figure 6.5.*

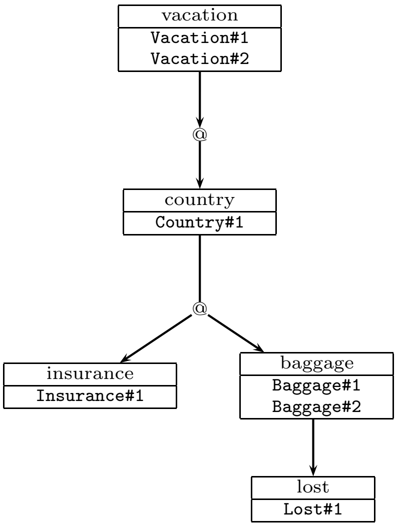


Figure 6.4: Part of the semantic ACH associated to the context of `Insurance and lost baggage` before semantic filtering

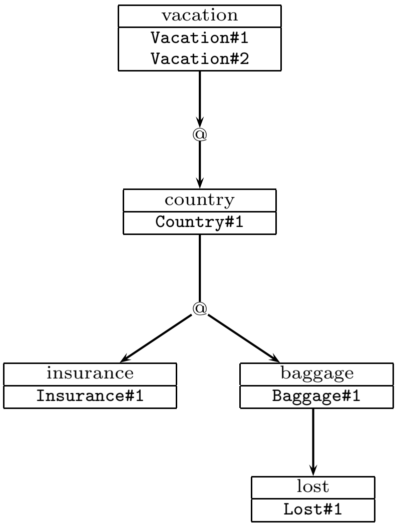


Figure 6.5: Part of the semantic ACH associated to the context of `Insurance and lost baggage` after semantic filtering

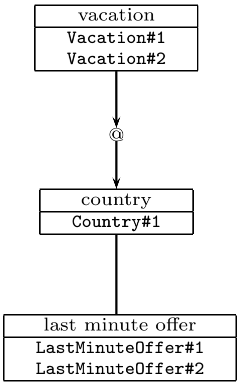


Figure 6.6: Part of the semantic ACH associated to the context of `Last minute offers` before semantic filtering

E_{PT}	E_{ACH}
⟨`Vacation#1`, @, ⟨`location#1`⟩⟩	⟨`LastMinuteOffer#1`, `Vacation#1`, ⟨⟩⟩
⟨`Vacation#2`, @, ⟨`location#1`⟩⟩	⟨`LastMinuteOffer#2`, `Vacation#1`, ⟨`about#1`⟩⟩
⟨@, `Country#1`, ⟨`location#1`⟩⟩	

Table 6.2: E_{PT} and E_{ACH} sets

Example 12 *Suppose to consider the node* `Last minute offers` *of Figure 2.1 and the corresponding context:* `Vacations`*. For the sake of simplicity, we only consider the attributes of the node* `Vacations`*. Suppose that the function* PARSE–TREE *works on the node* `Vacations` *as described in Section 6.2 and that it associates the label 'last minute offers' to the following two concepts:*

1. `LastMinuteOffer#1`*: a vacation proposal which occurs just before the departure date with advantageous economic conditions;*

2. `LastMinuteOffer#2`*: an offer of something which occurs just before a deadline is off.*

Moreover, suppose that the function SEMANTIC–ENRICHMENT *has provided two relations: an* IsA *from* `LastMinuteOffer#1` *to* `Vacation#1` *and a relation* `About#1` *from* `LastMinuteOffer#2` *to* `Vacation#1`*. Figure 6.6 represents the concepts associated to each parse tree element.*

We can summarize the composition of the sets E_{PT} *and* E_{ACH} *as in Table 6.2 (we omit the ACH nodes and the parse tree nodes in the triples).*

Heuristic rules applied locally do not alter the composition of the semantic ACH. Global filtering considers the headers of the parse trees: we can identify the presence of an IsA *relation between* `LastMinuteOffer#1` *and* `Vacation#1` *which makes it so that any other sense of 'last minute offer' and 'vacation' is removed for the* subsumption rule*. Note that this process removes also all the*

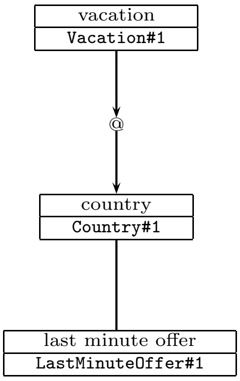


Figure 6.7: Part of the semantic ACH associated to the context of `Last minute offers` after semantic filtering

relations involving the concepts deleted, i.e. the second relation in E_{PT} and in E_{ACH}. The remaining concepts are shown in Figure 6.7.

6.5 Choosing an interpretation

We have claimed that the process of semantic filtering is not able to fully disambiguate the meaning of each word in the ACH and the way concepts are semantically related. Node interpretations can be arbitrarily complex but we expect that an interpretation of an ACH node (by human beings) is subject to some widely applied rules. For example, the concept `Insurance#1` in Example 11 could be semantically related to the concept `Vacation#1` through either the role `about#1` or the inverse of `organizedBy#1`. Probably, a human will not consider both the possibilities but it will only select one of the two: either 'insurance (policy) for vacations' or 'vacations organized by an insurance agency'. Since our goal is to reproduce as accurately as possible the human behaviour, we need to capture the empirical rules that make a *node interpretation* unequivocal. In other words, we aim at identifying all and only the concepts and relations which are essential for the meaning of a certain node. Such concepts and relations represent the *interpretation* of the node though they have not been expressed and assembled in a DL term yet.

Definition 19 (Node coverage) *Let H_k be the context of the node k in the ACH H and $H_{s,k} = \langle H, PT, E_{ACH}, E_{PT}, \mathrm{lab}_s\rangle$ the restriction of the semantic ACH H_s on H_k. A set $E \subseteq \{E_{ACH} \cup E_{PT}\}$ is said to be a* coverage *for the node k, if it satisfies the following conditions (e and e' represent generic elements of E):*

1. *E is acyclic;*

2. *$\forall e\, \forall e'(e = e^{ACH}_{\langle k,\_,\_\rangle \leftarrow} \wedge e' = e^{ACH}_{\langle k,\_,\_\rangle \leftarrow} \Rightarrow (\boldsymbol{R}_e = \boldsymbol{R}_{e'}) \wedge \exists k'(e = e^{ACH}_{\langle k',\_,\_\rangle \rightarrow} \wedge e' = e^{ACH}_{\langle k',\_,\_\rangle \rightarrow}))$;*

3. *$\forall e\, \forall e'(e = e^{PT}_{\langle k,n,\_\rangle \leftarrow} \wedge e' = e^{PT}_{\langle k,n,\_\rangle \leftarrow} \Rightarrow \boldsymbol{R}_e = \boldsymbol{R}_{e'})$;*

4. *$\forall e(e = e_{\langle k,n,s\rangle \rightarrow} \Rightarrow \exists e'(e' = e_{\langle k,n,s\rangle \leftarrow}) \vee \neg \exists e'(e' = e_{\langle k,n,\_\rangle \leftarrow}))$.*

Condition 1 prevents the semantic relations from being cyclic. Note that this constraint is applied only to global relations, because local ones, e in E_{PT}, are mono–directional (see Algorithm 6.2). Given two ACH nodes, condition 1 avoids, for example, that the corresponding meanings are both subject and modifier mutually. Condition 2 ensures that there are not two ACH relations, e in E_{ACH}, converging to the same ACH node unless they come from the same node and they refer to the same semantic relation. Intuitively, it means that a node can be modifier of another ACH node at most. In this way, we also prevent two nodes from being related by different semantic relations. Condition 3 is similar to condition 2 but it refers to the local case: a modifier and the corresponding subject can be linked by one semantic relation at most. Finally, condition 4 ensures that the meaning of each parse tree element (including headers) stays consistent. Intuitively, it means that if we have associated a certain concept to a certain parse tree element (in order to link it with its subject), we cannot change the element meaning (i.e. choosing another concept) in order to link the element with its modifier(s). As an example, imagine a parse tree element containing the word 'bank'. We cannot choose the sense 'bank as credit institute' to semantically relate it with its subject, and then choose the sense 'bank as part of a river' to relate it with its modifier(s).

Example 13 *Consider the node* `Insurance and lost baggage` *of Example 11 and the corresponding semantic ACH. A possible* coverage *(E_1) is represented by the following set of relations:*

⟨`Vacation#1`, @, `location#1`⟩

⟨@, `Country#1`, `location#1`⟩

⟨`Insurance#1`, `Vacation#1`, `about#1`⟩

⟨`Baggage#1`, `Lost#1`, `state#3`⟩

Another possible coverage *(E_2) is given by the following set:*

⟨`Vacation#1`, @, `location#1`⟩

⟨`Vacation#2`, @, `location#1`⟩

⟨@, `Country#1`, `location#1`⟩

⟨`Vacation#1`, `Insurance#1`, `organizedBy#1`⟩

⟨`Baggage#1`, `Lost#1`, `state#3`⟩

They mainly differ for the selection of the semantic relation connecting the concepts `Vacation#1` *and* `Insurance#1`*. Moreover, in the first case, we cannot consider the edge* ⟨`Vacation#2`, @, `location#1`⟩ *because* `Vacation#2` *is not the* *modifier of any other concept and the other sense of 'vacation' has such a role (see condition 4 of Definition 19).*

Note that the empty set is a possible coverage *(E_3), too, since all the conditions are automatically satisfied.*

From Definition 19 and from the above example, we can easily deduce that a set of semantic relations can generate more possible *node coverages*, i.e. interpretations. Formally, it exists one possible *coverage*, only if no semantic relations have been found (i.e. $E_{ACH} = E_{PT} = \emptyset$). Indeed, if one semantic relation has been found, at least two possible coverages exist, the one considering no relations at all, and the one considering the relation found.

Let us call $\Gamma \subseteq 2^{\{E_{ACH} \cup E_{PT}\}}$ the set of all the possible *coverages* of a given node k. In the following, we propose three heuristics in order to choose one (or a subset) of them. As a support, we first introduce some further definitions.

Definition 20 (Senses induced by a node coverage) *Let $H_{s,k} = \langle H, PT, E_{ACH}, E_{PT}, \mathrm{lab}_s \rangle$ be the semantic labeling of the context of a node k and $E \in \Gamma$ a possible coverage for k. The set of* senses induced by *E, $I(E)$, is defined as follows:*

$$I(E) = \{s \mid \exists k \in H \ \exists n \in PT(k) \ (s \in \text{Senses}(n) \wedge (\exists e \in E \\ (e = e_{\langle k,n,s \rangle \leftrightarrow}) \ \vee \ \neg\exists e \in E(e = e_{\langle k,n,\_ \rangle \leftrightarrow}))))\}$$

The senses induced by a node coverage are those which are involved as antecedents or subsequents of its relations. If a parse tree node is neither the starting point nor the arrival point of any relation, we select all the senses of that parse tree element.

Definition 21 (Taxonomic relations) *Let $H_{s,k} = \langle H, PT, E_{ACH}, E_{PT}, \mathrm{lab}_s \rangle$ be the semantic labeling of the context of a node k and $E \in \Gamma$ a possible coverage for k. The set of* taxonomic relations *in E, $T(E)$, is defined as follows:*

$$T(E) = \{e \in E \mid \exists k \ \exists k' \in H \ (e = e^{ACH}_{\langle k,\_,\_ \rangle \rightarrow} \ \wedge \ e = e^{ACH}_{\langle k',\_,\_ \rangle \leftarrow} \ \wedge \\ k' \in DESCENDANT(k))\}$$

where $DESCENDANT : K \rightarrow 2^K$ is a function that relates a node k in H to the set of its descendants with respect to the hierarchy $\langle K, E \rangle$.

We define as taxonomic the global relations which have the same direction of the edges of an ACH structure. The edges of an ACH structure point from a node to all its children. Intuitively, they are called *taxonomic* because they consider a descendant node as a modifier of an its ancestor, therefore a specification of its concept.

We now present three heuristics to filter the set of possible coverages $\Gamma = \{E_1, E_2, \ldots, E_n\}$ of a certain node k.

Maximization of semantic relations:

$$\Gamma_M = \{E \in \Gamma \mid card(E) = max\{card(E_1), \ldots, card(E_n)\}\}.$$

Minimization of senses:

$$\Gamma_m = \{E \in \Gamma \mid card(I(E)) = min\{card(I(E_1)), \dots, card(I(E_n))\}\}.$$

Taxonomy biasing:

$$\Gamma_{tb} = \{E \in \Gamma \mid card(T(E)) = max\{card(T(E_1)), \dots, card(T(E_n))\}\}.$$

The first rule prefers the coverages with the largest amount of relations. Intuitively, it selects the interpretations which best connect the concepts in the structure. The second rule selects the coverages which involve the smallest number of senses. In other words, it prefers those which best disambiguate the meaning of the nodes. Since all the senses of a parse tree element are considered if the element is not related to the rest of the context, the minimization of the senses generally implies the maximization of the relations and vice versa. Finally, the third rule prefers the coverages with the largest number of relations connecting the ACH nodes to their descendants.

Example 14 *Let us consider Example 13 and three of the possible coverages of the node* `Insurance and lost baggage`*, namely E_1, E_2 and E_3. The following table shows the corresponding number of relations, of senses induced and of taxonomic relations.*

	card	*I*	*T*
E_1	*4*	***5***	*0*
E_2	***5***	*6*	***1***
E_3	*0*	*6*	*0*

E_1 minimizes the number of senses, whereas E_2 maximizes both the number of semantic relations and of taxonomic relations. The final choice of one coverage for the node `Insurance and lost baggage` *depends on the priority which is assigned to each heuristic.*

Algorithm 4.1 uses the function CHOOSE–COVERAGE to select one coverage among those which are allowed by the semantic information available. It takes a semantic ACH $H_s = \langle H, PT, E_{ACH}, E_{PT}, lab_s \rangle$ representing the context (enhanced with semantic information) of a given node k and it selects one of the possible coverages E in Γ, according to one (or more) heuristic criterion (and possibly randomly). Moreover, it discards all the relations in $\{E_{ACH} \cup E_{PT}\}$ which do not belong to the coverage E, and all the senses in PT which are not *induced* by E. It finally returns the resulting semantic ACH containing only the senses and the relations in E.

6.6 Composing the meaning

The previous steps of semantic explicitation have led to gather the essential and unequivocal information for the interpretation of a given node: a set of senses denoting the meaning of each element in the context of the node and a set of relations holding between such concepts. In this section, we intend to express and formalize the *interpretation* (represented by a *coverage*) in Description Logic.

When combining different elements in a logic formula, we need to choose the way of expressing the intended global concept with the fragment of the DL framework we are going to consider. As already claimed, we employ the $\mathcal{ALCO}$ fragment for what concerns concept constructors, while the choice about role constructors depends on the roles returned by the function $\mathcal{SR}_{\mathcal{O}}$. Obviously, the decision is not free from relevant consequences, since it strongly impacts on the algorithm decidability and efficiency.

In general, we express that the concept M is a modifier of the concept S through the semantic relation $\mathbf{R} = \langle \mathsf{R}_1, \mathsf{R}_2 \ldots, \mathsf{R}_n \rangle$ ($\mathbf{R} \neq \emptyset$) with a DL existential restriction as follows:

$$\mathsf{S} \sqcap \exists \mathsf{R}_1 . \exists \mathsf{R}_2 \ldots \exists \mathsf{R}_n . \mathsf{M}$$

We will use the above expression when linking different elements of the same ACH node or the concepts which refer to different nodes in the context of the one we are examining. In case $\mathbf{R} = \langle R \rangle$ is a single–element set, the term reduces to:

$$\mathsf{S} \sqcap \exists \mathsf{R} . \mathsf{M}$$

We express that two concepts S_1 and S_2 are coordinate (i.e. each of them is independent from the other) by means of a disjunction term:

$$\mathsf{S}_1 \sqcup \mathsf{S}_2$$

The expression is needed in those cases where node labels have more subjects (e.g. 'insurance and lost baggage') or where attribute fillers are sets of values (i.e. $F = F_f$). In both cases, the alternatives (i.e. different subjects and different attribute values, respectively) are linked with the disjunction operator.

In Algorithm 6.6, we propose the algorithm to compose the meaning. It takes the semantic ACH (containing one coverage) referring to a node k and returns the corresponding *node interpretation*, i.e. a concept term.

The algorithm has two distinct parts: the former (steps 1–4) builds the DL terms expressing the local meaning of each node in the context, while the latter (steps 5 and 6) assembles them and produces the final interpretation. The local interpretations are generated by the function BUILD–LOCAL–MEANING and stored in an array, F, indexed by the nodes. The input of BUILD–LOCAL–MEANING is composed by three elements: the parse tree of the current node k, the corresponding root (see step 3) and the set of relations connecting the elements of k (see step 2). Step 5 collects in M all the ACH nodes, m, which are not modifiers of any other node, i.e. they are not subsequents of any relation in the coverage. Step 6 builds the final term by arranging the DL formulae representing the context subjects, m (and their modifiers). Note that if the coverage is a connected graph (all the nodes are related), the context has one subject only (i.e. the cardinality of M is 1). Thus, we use the conjunction operator of step 6 only if we have not been able to relate all the nodes in

```
Algorithm 6.6 COMPOSING–THE–MEANING($H_s$)
    ▷ semantic ACH: $H_s = \langle H, PT, E_{ACH}, E_{PT}, \text{labs}_s \rangle$
    VarDeclarations
        set of relations: $E_k$
        parse tree node: $n$
        array of DL terms: $F$
        set of ACH nodes: $M$
        DL concept term: $t$
    for each node $k \in H$
        $E_k \leftarrow \{e \mid e = e^{PT}_{\langle k,\_,\_\rangle \leftrightarrow}\}$;
        $n \leftarrow \mathrm{ROOT}(PT(k))$;
        $F(k) = $ BUILD–LOCAL–MEANING$(PT(k), n, E_k)$;
    $M \leftarrow \{k \in H \mid \neg\exists e\ (e = e^{ACH}_{\langle k,\_,\_\rangle \leftarrow})\}$;
    $t \leftarrow \bigsqcap_{m_i \in M}$ BUILD–GLOBAL–MEANING$(H, F, m_i, E_{ACH})$;
    Return $t$;
```

the context. In this case, we assume the DL constructor ⊓ as default logical connection between the subjects. For example, consider a node `Vacations` representing the concept Vacation and its context Insurance representing the concept `Insurance`. Moreover, suppose that a possible coverage for the latter node does not include any relation between them. In this case, the interpretation of `Vacations` would be:

$$\mathsf{Vacation} \sqcap \mathsf{Insurance}$$

We now analyze Algorithm 6.7 which builds the meaning of a single ACH node. The algorithm works in a recursive way by building the term of the input node and connecting it to the one of its modifiers. The input elements are a parse tree, a set of relations between its elements and one of its nodes. In steps 1 and 2 we deal with the virtual nodes, @. They are used within a node label either when it has more *headers* or to introduce an attribute. In the former case, the children of @ are the different subjects, while in the latter case they are the possible attribute values (see the Algorithm 6.1). In both cases, the children are required to be in disjunction as the following examples show.

Example 15 *A label as 'insurance and baggage' with two subjects, 'insurance' and 'baggage', will be interpreted as:*

$$\mathit{Insurance} \sqcup \mathit{Baggage}$$

where Insurance *and* Baggage *are assumed to be two concepts of a given ontology.*

Example 16 *Let us consider the node* `Last minute offers` *of Figure 2.1 and its attribute* price*:* $\langle price, \{100\$, 500\$, 1000\$\}\rangle$*. In order to express the possible values of the filler, we will write:*

$$\{\mathit{100\$}\} \sqcup \{\mathit{500\$}\} \sqcup \{\mathit{1000\$}\}$$

```
Algorithm 6.7 BUILD–LOCAL–MEANING$(P, n, E)$
      ▷ parse tree: $P$
      ▷ parse tree node: $n$
      ▷ set of relations: $E$
    VarDeclarations
        DL term: $t$
        array of semantic relations: $A$
1   if $(n = @)$
2       Return $(\bigsqcup_{c \in CHILD(n)}$ BUILD–LOCAL–MEANING $(P, c, E))$;
3   $t \leftarrow (\bigsqcup_{s \in Senses(n)} s)$;
4   for each $c \in CHILD(n)$
5       if $(\exists e \in E\ (e = e_{\langle \_, n, \_ \rangle \rightarrow}\ \wedge\ e = e_{\langle \_, c, \_ \rangle \leftarrow}))$
6           $A(c) = \mathbf{R}_e$;
7       else $A(c) = \mathbf{R}^*$;
8   $t \leftarrow (t\ \sqcap\ \bigsqcap_{c \in CHILD(n)} \exists A(c).$ BUILD–LOCAL–MEANING$(P, c, E))$;
9   Return $t$;
```

where $\{\textsf{100\$}\}$, $\{\textsf{500\$}\}$ *and* $\{\textsf{1000\$}\}$ *are assumed to be nominals of a given ontology.*

Step 3 produces the DL term of the input node. We combine all its possible meanings (i.e. those which are induced by the coverage) through the disjunction operator. Actually, we have more concepts in disjunction only if we have not been able to fully disambiguate the meaning of the parse tree element. Steps 4–7 identify the semantic relation holding between the current node and its children (i.e. its modifiers) and store it in the array A (step 6). Note that from Definition 19, if more than one relation exists between the same elements, e.g. e_1 and e_2, they must refer to the same semantic relation, i.e. $\mathbf{R}_{e_1} = \mathbf{R}_{e_2}$. When we cannot identify a relation between an element and one of its modifiers, we apply a special role, $\mathbf{R}^*$, whose purpose is to be a placeholder for maintaining the dependency between the two elements according to the parse tree structure[2]. Step 8 connects the DL term of the current node with those of its modifiers by means of the previously set semantic relations. $A(c) = \mathbf{R} = \langle \mathsf{R}_1, \mathsf{R}_2, \ldots, \mathsf{R}_n \rangle$ is actually expanded as follows[3] (see the discussion at the beginning of the section): $\mathsf{R}_1.\exists\mathsf{R}_2 \ldots \exists\mathsf{R}_n$. Finally, note that all the modifiers are in conjunction one with each other.

Example 17 *Suppose that in Example 14, we prefer the coverage which maximizes the number of senses induced:* E_1. *The function* BUILD–LOCAL–MEANING

[2] In semantic comparison, its semantics will be *any relation*. For example, consider the following three terms: (t_1) $\mathsf{C} \sqcap \exists\mathsf{R}.\mathsf{D}$, (t_2) $\mathsf{C} \sqcap \exists\mathsf{R}_1.\exists\mathsf{R}_2.\mathsf{D}$ and (t_3) $\mathsf{C} \sqcap \exists\mathsf{R}^*.\mathsf{D}$. In this case, we would assert that $t_3 = t_1$ and $t_2 = t_1$ but not necessarily $t_1 = t_2$.

[3] If $\mathbf{R} = \langle\rangle$, it represents the relation *IsA*. In this case, we simply connect the two concepts by means of the conjunction operator. For example, given two (complex) concepts C and D, we return $\mathsf{C} \sqcap \mathsf{D}$.

```
Algorithm 6.8 BUILD–GLOBAL–MEANING$(H, F, k, E)$
    ▷ ACH: $H = \langle K, E, \mathrm{att}, \mathrm{lab}\rangle$
    ▷ array of DL terms: $F$
    ▷ ACH node: $k$
    ▷ set of relations: $E$
    VarDeclarations
        set of ACH nodes: $C$
        relation: $e_{k\rightarrow c}$
        array of semantic relations: $A$
        DL term: $t$
1   $C \leftarrow \{c \in K \mid \exists e \in E\ (e = e_{\langle k,\_,\_\rangle\rightarrow}\ \wedge\ e = e_{\langle c,\_,\_\rangle\leftarrow})\}$;
2   for each $c \in C$
3       $e_{k\rightarrow c} \leftarrow e \in E\ (e = e_{\langle k,\_,\_\rangle\rightarrow}\ \wedge\ e = e_{\langle c,\_,\_\rangle\leftarrow})$;
4       $A(c) \leftarrow \mathbf{R}_{e_{k\rightarrow c}}$;
5   $t \leftarrow (F(k)\ \sqcap\ \bigsqcap_{c\in C} \exists A(c).$BUILD–GLOBAL–MEANING$(H, F, c, E))$;
6   Return $t$;
```

returns the following DL term for the node `Vacations`*:*

`Vacation#1 ⊓ ∃location#1.Country#1`

For the header 'insurance' of the node `Insurance and lost baggage`*, it returns:*

`Insurance#1`

Whereas the DL term associated to the header 'baggage' is:

`Baggage#1 ⊓ ∃state#3.Lost#1`

Finally, step 2 connects these two headers through the disjunction operator:

`(Insurance#1) ⊔ (Baggage#1 ⊓ ∃state#3.lost#1)`

Finally, we present BUILD–GLOBAL–MEANING in Algorithm 6.8. Its goal is to connect the meaning of the nodes in the context to build the contextual interpretation of a given element.

Algorithm 6.8 takes a set of relations, E, through which contextual nodes are connected. Furthermore, it is provided with the DL terms, F, representing the local interpretation of each of these nodes. The process is recursive: given a node k as input, the algorithm combines its term, $F(k)$, with the one of its modifiers (step 5). Step 1 identifies all the modifiers of the current node according to the coverage E, whereas steps 2–4 store the corresponding semantic relation in the array A. The way of treating $A(c)$ is the same as in BUILD–LOCAL–MEANING. Step 6 returns the resulting complex concept. Note that the procedure is very similar to Algorithm 6.7.

Example 18 *With reference to the coverage E_1 of Example 14 and to the local interpretations of Example 17,* BUILD–GLOBAL–MEANING *combines the meaning of the nodes* `Vacations` *and* `Insurance and lost baggage` *as follows:*

$$((\texttt{Insurance\#1}) \sqcup (\texttt{Baggage\#1} \sqcap \exists\texttt{state\#3.lost\#1})) \sqcap$$
$$\exists\texttt{about\#1}.(\texttt{Vacation\#1} \sqcap \exists\texttt{location\#1.Country\#1})$$

It represents the formalization in Description Logic of the coverage E_1, i.e. the node interpretation *of* `Insurance and lost baggage` *contextualized with the node* `Vacations`*.*

Chapter 7

Implementation

7.1 Process flow and technologies

One of the main goals of our investigations was the development of a schema matching system based on a semantic method. Our intent was the evaluation of the algorithm precision in computing the meaning of the schema elements compared to the previous version, CtxMatch, and the comparison with the results of traditional methodologies for schema matching. Hence, we have implemented a prototypal application for the experimentation of our theories on simple test cases, whereas the development of a full–featured system for performance comparison represents the work of the next future.

In this section, we analyze the whole process of schema matching, focusing on the system components and on the corresponding interactions. We will explain the technologies we have employed for their realization and the external resources which support the system. In the next section, we will deal in detail with some aspects which depend on the implementation technology adopted.

We can identify two main components in the architecture of the system. They correspond to the two basic steps of semantic methods: semantic explicitation and semantic comparison.

We start by analyzing the first component. Figure 7.1 shows the corresponding process flow. The first process is the *context extractor* which corresponds to the function CONTEXT in Algorithm 4.1. It works on the input ACH which we implement as an XML document. The browsing and the processing of the tree structure of the ACH is performed with the help of the Saxon technology by Michael Kay [12]. While working on a Java application, it can be accessed from it by using the standard JAXP API. The context of the node in exam is then passed to the process of *tree parsing* (corresponding to the function PARSE TREE in Algorithm 4.1), which parses the labels and the attributes of the nodes. We use MiniPar [6], a broad–coverage parser for the English language, as default tree parser (it corresponds to the function NLP–PARSING in Algorithm 6.1). The next step is the *lexical enrichment* of parse trees, i.e. the association of each word (or

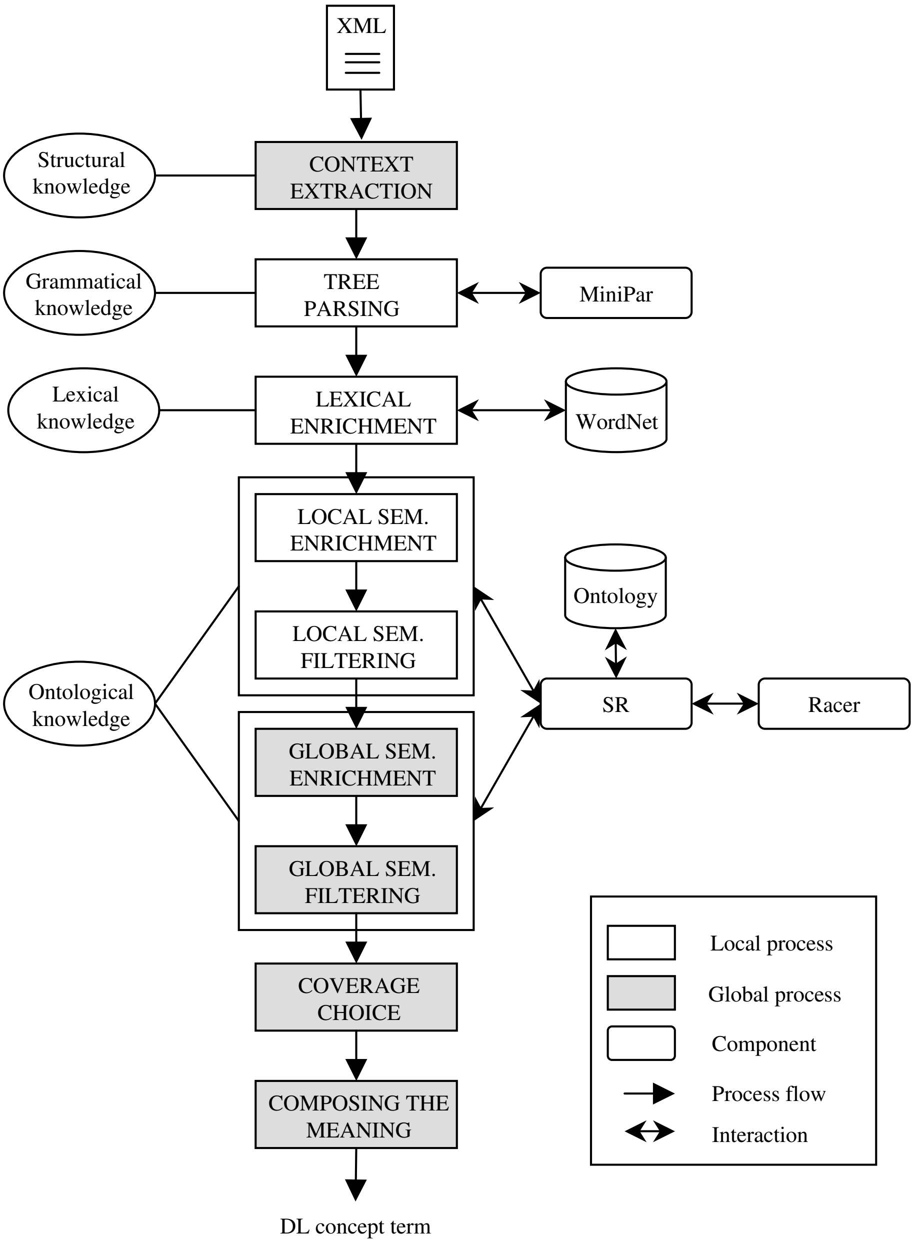


Figure 7.1: Semantic explicitation processes and interaction with external resources

multi–word) with the corresponding lexicon. We use the WordNet dictionary [16] as *lexicon* function (see Definition 14). Semantic explicitation continues by enriching the ACH with semantic relations and by filtering the possible word senses. *Semantic enrichment* and *semantic filtering* are first performed *locally* (i.e. independently from the contextual nodes) and then *globally*. The extraction of semantic relations from existing ontologies is executed by the component *SR*. It corresponds to the homonymous function (see Definition 13) and it has been implemented according to the algorithm proposed in Section 5.3 (see the next section for implementation details). It derives semantic relations between ontological concepts by interacting with RACER [11] (or Pellet [10]), a Description Logic reasoning system by Volker Haarslev and Ralf Mller. We use OWL DL [15] (or equivalent languages, e.g. DAML, RACER language) ontologies as supporting knowledge bases. However, they present one main drawback: concepts and roles in ontologies are usually expressed as strings of natural language, whereas WordNet returns senses. Thus, passing from WordNet synsets to ontological concepts (and roles) and vice versa is not trivial[1]. At the current time, we have solved the problem by simply tagging the ontologies of interest with WordNet senses, manually. This problem does not exist if we directly use the knowledge contained in WordNet. Thus, for practical purposes, we have also used the linguistic relations which WordNet makes available (e.g. hypernymy, hyponymy, meronymy, holonymy) by assigning an ontological value to them[2]. So, for example, hyponymy has been mapped onto the *IsA* semantic relation, holonymy onto the *PartOf* relation, etc.

The process of semantic explicitation ends by *choosing* one possible *coverage* for the node in exam and by representing it through the process *composing the meaning*.

We point out that those processes which are depicted in grey in Figure 7.1 work in a global perspective with the aim of contextualizing the local meaning of the node in exam. These also represent those steps which must be performed independently for each node we need to interpret. On the contrary, white boxes are processes acting locally, i.e. node by node, thus they can be computed all in one batch for each node of the ACH and then be re–used in the semantic explicitation of any schema element. Finally, Figure 7.1 presents the main type of knowledge required by each step of the process on the left.

The second main component of the schema matching system is *semantic comparison* whose goal is to compute the semantic relation existing between two (or a set of) DL concept terms. Figure 7.2 presents an overall view of the system with particular emphasis on this component (in grey color). Its input is a pair of DL concept terms. Each term is the result of the computation performed by some *semantic explicitation* process. We can imagine that XML1 and XML2 in Figure 7.2 are two independent schemas (sited at any geographical distance) and each process of semantic explicitation produces the corresponding interpretation. Note that the external resources (e.g. lexicon, knowledge bases)

[1] We discuss this problem and the possible ways of solving it in Chapter 9.

[2] See Chapter 9 for a more formal discussion about this issue.

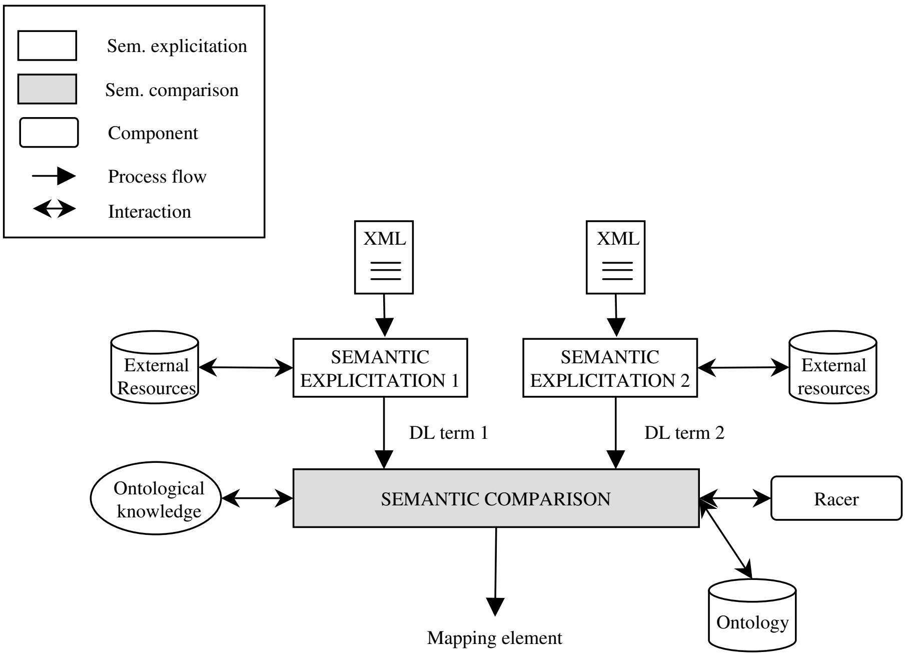


Figure 7.2: Semantic comparison process and interaction with external resources

which each process uses are not required to be the same (e.g. each party can employ its own background knowledge). The component *semantic comparison* compares the results of the two mentioned processes and identifies the logical relation between the terms. It is based on DL inferential procedures (see Section 4.2.3), thus it needs the support of a DL reasoner interacting with a background knowledge base. Note that *semantic comparison* and its external resources are independent from the processes and the resources which have produced the interpretation of the two schemas to be matched. The decidability of the algorithm of semantic comparison is ensured by the employment of ontologies in OWL DL which return roles guaranteeing the decidability of our logic framework (see Section 4.2.3).

7.2 Java components

The need to integrate and interact with numerous existing technologies (e.g. Saxon, RACER, tree parsers) whose implementation or API is in Java represents the main cause which has led us to the employment of the same technology for developing our prototypal application.

In this section, we analyse in more detail two specific aspects of the system architecture: how the *semantic ACH* data structure has been realized and how the component *SR* has been implemented.

The *semantic ACH* presented in Section 6.1 reproduces the tree structure of the corresponding ACH and stores the different types of information which are needed to interpret the schema. It stores (i) the grammatical structure and the lexicon of the ACH labels (and attributes) in *parse trees* (see the *PT* function), (ii) the *semantic relations* holding between ACH concepts in E_{PT} and E_{ACH} and (iii) the concept terms expressing the *node interpretations* in lab_s.

Figure 7.3 shows the implementation details of the *semantic ACH.* Let us focus on the classes *Context* and *ContextElement* which realize the structure of the ACH as a *tree. Context* represents the ACH. It contains the root node (represented by the variable *root*) which in turn is linked to all its children (represented by the variable *children*). Each node k of the ACH is implemented as an object belonging to the class *ContextElement.* It is provided with a set of variables as follows:

- an identification number *id*;
- the path of the node in the ACH *xPath*;
- the node label (i.e. $lab(k)$) *label*;
- the identification number of the father *fatherId*;
- the link to its children (i.e. E) *children*;
- the node interpretation (i.e. $lab_s(k)$) *globalConcept.*

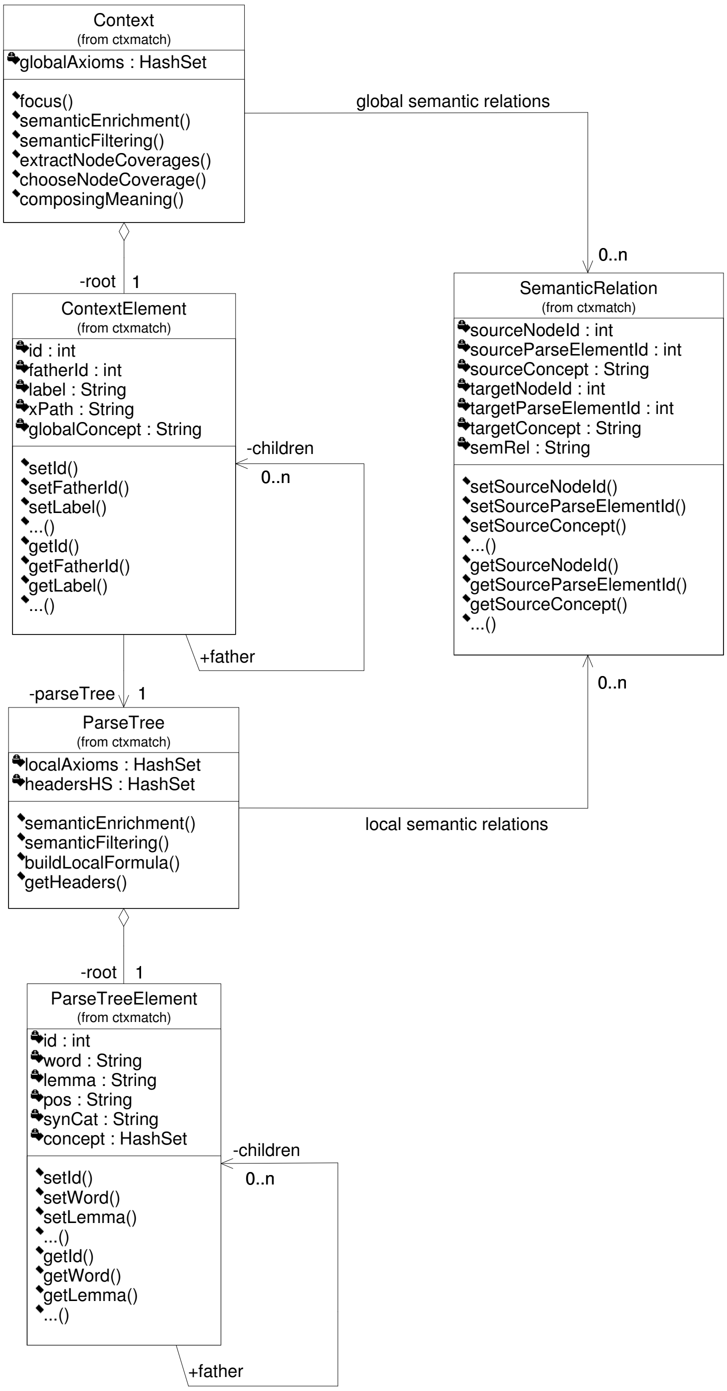


Figure 7.3: Class diagram highlighting the semantic ACH structure

The link to the children is allowed by means of Java Hash Tables both to avoid the introduction of an order among them (as in the case of Vectors or Arrays) and for the sake of an efficient and easy access to the single elements of the set. Note that in the current implementation, attributes are not contemplated yet. The methods of *ContextElement* are the typical *get* and *set* functions defined for each variable. Each element of the type *ContextElement* is associated to the corresponding parse tree (due to the *PT* function) implemented as *ParseTree*. The relation existing between *ParseTree* and *ParseTreeElement* is the same as between *Context* and *ContextElement*, i.e. *ParseTree* represents the whole parse tree by containing the parse tree root (represented by the variable *root*) which in turn is linked to all its children (represented by the variable *children*). Indeed, remember that both ACH and parse trees are organized as trees. Each parse tree node is implemented as an object of the class *ParseTreeElement*. A *ParseTreeElement* is composed by:

- an identification number *id*;
- the word it represents (i.e. *Word*) *word*;
- the word lemma (i.e. *Lemma*) *lemma*;
- the word part of speech (i.e. *POS*) *pos*;
- the word syntactical category (i.e. *SynCat*) *synCat*;
- the word lexicon (i.e. $\mathcal{L}_O$ applied to *Word*) *concept*;
- The link to its children *children*.

Even in this case, the access to the children of the element and to the word lexicon occurs by means of Hash Tables, and the class methods are the typical *get* and *set* functions.

An important variable of the class *ParseTree* is *headerHS* which keeps track of all the *header* elements of the parse tree (i.e. the subject(s) of the corresponding label).

Relations in E_{PT} and E_{ACH} are represented by the class *SemanticRelation*. Each element of such sets is composed by a triple as follows: $\langle\langle k_A, n_A, s_A\rangle, \langle k_B, n_B, s_B\rangle, \mathbf{R}\rangle$. The terms of the relation are mapped 1:1 onto the variables of the class *SemanticRelation*:

- k_A onto *sourceNodeId*;
- n_A onto *sourceParseElementId*;
- s_A onto *sourceConcept*;
- k_B onto *targetNodeId*;
- n_B onto *targetParseElementId*;

- s_B onto *targetConcept*;
- **R** onto *semRel*.

Note that the semantic relation **R** is here represented as a string, i.e. the concatenation of the names of the roles composing **R**. The relations in E_{PT} are stored in the Hash Set *localAxioms* of the class *ParseTree*, whereas the relations in E_{ACH} in the Hash Set *globalAxioms* of the class *Context*.

Context and *ParseTree* are provided with all those methods implementing the algorithms of semantic explicitation as presented in Chapter 6. *Context* methods work at global level (i.e. with respect to all the contextual nodes), whereas local operations (i.e. working on a single ACH node) are directly performed by *ParseTree*. With reference to the process flow described in the previous section, we can recognize the methods which realize the single processes, respectively:

- *focus* in *Context* extracts the context of a given ACH node;
- parse trees and the corresponding lexicon are automatically built when defining a new *ContextElement* (in the class constructor);
- *semanticEnrichment* in *ParseTree* performs local semantic enrichment;
- *semanticFiltering* in *ParseTree* performs local semantic filtering;
- *semanticEnrichment* in *Context* performs global semantic enrichment;
- *semanticFiltering* in *Context* performs global semantic filtering;
- *extractNodeCoverages* and *chooseNodeCoverage* in *Context* choose one node coverage;
- *buildLocalFormula* in *ParseTree* builds the concept term expressing the interpretation of a single ACH node;
- *composingMeaning* in *Context* builds the concept term expressing the contextual interpretation of a given ACH node.

All the methods reproduce the corresponding algorithm presented in Chapter 6. For the sake of relevance from an implementation point of view, we point out that the procedures for building local formulas and for composing contextual interpretations are *recursive* algorithms.

We finally present the architecture of the component *SR* which extracts semantic relations from existing ontologies, i.e. implementing the algorithm of Section 5.3. Figure 7.4 shows the classes involved. The main class is *ExtractRelation*. It is fed with an ontology and provides some functionalities to explore it. It can identify all the concept names (i.e. N_C) and the role names (i.e. N_R) by means of the methods *getConcepts* and *getRoles*, respectively. The core method is *getSemanticRelation* which corresponds to the function $\mathcal{SR}$. It needs

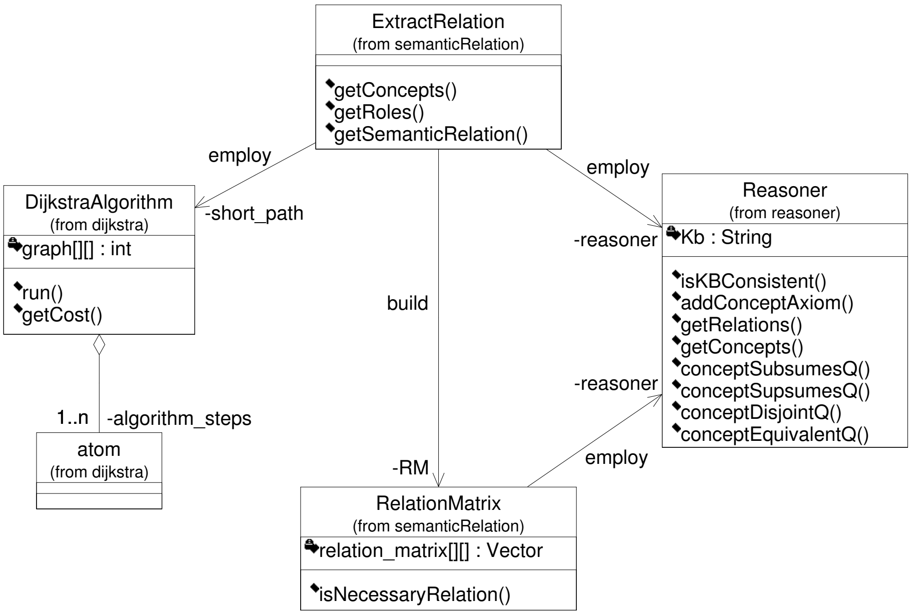


Figure 7.4: Classes involved in the extraction of semantic relations from ontologies

three more classes in order to work properly: *Reasoner*, *RelationMatrix* and *DijkstraAlgorithm*.

The first class allows to edit and make inference on the given ontology (represented by the variable *Kb*) by the interaction with a DL reasoner (e.g. RACER). The methods *addConceptAxiom* and *isKBConsistent* are used to add concept axioms to the ontology and to check its consistency, respectively. They are mainly used in step 6 of Algorithm 5.1 in order to check the consistency of the knowledge base after having derived and stated the possible presence of a certain semantic relation between two given concepts. Inference is performed by means of the methods *conceptSubsumesQ*, *conceptSupsumesQ*, *conceptDisjointQ* and *conceptEquivalentQ*, which search possible logical relations between two input concepts as it can be easily imagined by looking at the names of the methods.

Class *RelationMatrix* represents the homonymous data structure. It is implemented as a matrix of Vectors (represented by the variable *relation_matrix*), where each element of a Vector is a candidate for sematically relating the concepts represented by the matrix coordinates of the vector. *RelationMatrix* employs *Reasoner* in order to make inference as largely visible in Algorithm 5.2. We finally mention the method *isNecessaryRelation*: it allows to check whether a certain semantic relation between two given concept is *possible* or *necessary* (see Section 5.2).

Class *DijkstraAlgorithm* implements the Dijkstra algorithm for calculating the shortest path between two nodes in an oriented graph. The graph is realized as a matrix (represented by the variable *graph*) of integers (i.e. edge costs). Each step of the algorithm is represented by an object belonging to the class *atom*. We finally mention the method *getCost* which computes the cost of a given path between two nodes.

Chapter 8

Running example

In this section, we present one of the simple test cases we have dealt with. In this thesis we have not been able to perform complex and complete testing since it would have required a lot of extra–work. The implementation of an architecture to support intensive testing and to compare performances with other matching systems will be the object of future work.

One of the main limits of the practical part of the thesis has been the lack of ontologies being both *general* (i.e. describing a range of concepts widely shared) and *precise* (i.e. rich in information describing and constraining concepts). We believe that satisfying both requirements is very hard when considering non–specific domains as our analysis on some of the most popular ontologies (e.g. SUMO [13], OpenCyc [9], Mikrokosmos [5]) has confirmed. This is the reason which has led us to focus on specific domains and not on large–scale contexts like web directories. In specific domains satisfying these requirements is much simpler and some existing ontologies can effectively do this (see for example the DAML library [1]). Anyway, we are still far from having a large repository of well–founded knowledge bases: think for example that at the moment a large percentage of the ontologies in the DAML library are non–consistent as we have verified by means of RACER.

We now present our simple example tested on the prototypal architecture. It allows us to make a first check about the possible results of our methodology and to make some considerations concerning the use of the system in real cases.

Suppose to match the hierarchical classifications of Figure 8.1. In the following, we will refer to the left structure as H_1 and to the right as H_2.

First of all, we need to define the knowledge base which supports the process of semantic explicitation and comparison. We have not found any existing ontology providing a number of semantic relations between the terms of H_1 and H_2 (i.e. being general and precise with respect to the domain of interest). So, using the OWL DL language, we have created a small knowledge base, which formalizes concepts and relations in the *music* domain as in Table 8.1. We are aware of affecting the reliability of the test by creating the ontology on our own. Anyway, we think that no more significant axioms relating the possible

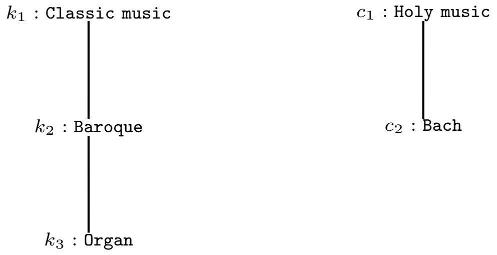


Figure 8.1: Classification structures for test case

TBox axioms
Music $\sqsubseteq$ $\exists$style.MusicStyle
Holy $\sqsubseteq$ MusicStyle
Classic $\sqsubseteq$ MusicStyle
Baroque $\sqsubseteq$ Classic
Music $\sqsubseteq$ $\exists$composed.Composer
Music $\sqsubseteq$ $\exists$played.(Instrument $\sqcup$ Voice)
Bach $\sqsubseteq$ Composer
Organ $\sqsubseteq$ Instrument
$\exists$style.Holy $\sqsubseteq$ $\exists$played.Organ
$\exists$composed.Bach $\sqsubseteq$ $\exists$style.Baroque

Table 8.1: Music domain ontology

meanings of the terms of H_1 and H_2 can be given[1]. Thus, the semantic relations which we are able to derive can be considered fairly complete with respect to the terms of the structures (and to the specific domain considered). Furthermore, the assertions of Table 8.1 seem reasonable and not biased towards our way of working.

The formalization given in Table 8.1 is the common way of expressing ontologies. However, as we have already claimed in Section 7.1, the use of WordNet as lexicon makes it necessary to map the ontological concepts and roles (expressed as natural language strings) on WordNet senses. Hence, we have manually tagged each term in the ontology with the corresponding senses as given in Table 8.2. A term as `example#Xn` means sense number n of the entry *example* of WordNet 2.0. X refers to the corresponding part of speech (N = noun, A = adjective, V = verb). The top index '-1' in case of verbs means that we refer to its passive form.

Once defined the knowledge base, we have started the process of semantic explicitation whose results are reported in the following tables:

- Tables 8.3 and 8.6 contain the results of those operations which are inde-

[1] We could assert, for example, that instruments are built according to some style (e.g. baroque). This would certainly introduce some more relations between the concepts in H_1, however, it would be a piece of information not strictly related to the music domain as we intend it here (i.e. not referring to the way instruments are built).

Terms	WordNet senses
Music	Music#N1, Music#N3
MusicStyle	Style#N3
Classic	Classic#A1
Baroque	Baroque#A1
Holy	Holy#A1
Instrument	Instrument#N6
Composer	Composer#N1
Voice	Voice#N2, Voice#N7
Organ	Organ#N5, Organ#N6
Bach	Bach#N1
style	style#N3
composed	compose#V2^{-1}
played	play#V3^{-1}

Table 8.2: Mapping ontological concepts and roles on WordNet senses

pendent from the node to be interpreted, i.e. *tree parsing*, *lexical enrichment*, *local semantic enrichment* and *local semantic filtering*. Table 8.3 refers to H_1, while Table 8.6 to H_2.

- Tables 8.4 and 8.7 contain the results of *global semantic enrichment* and *global semantic filtering*, i.e. contextual processes. The tables report the results once focused on the context of each node of H_1 and H_2, respectively.
- Tables 8.5 and 8.8 contain the results of the processes *choose coverage* and *composing the meaning* (both locally and globally). The heuristic applied for choosing the coverage has been the *maximization of semantic relations*. However the results would be the same as in case of *taxonomy biasing*. Table 8.5 refers to H_1, while Table 8.8 to H_2.
- Table 8.9 shows the matching results, i.e. the mapping elements.

Note that semantic relations are represented as pairs where the former term represents the direction of the relation (from subject to modifier) and the latter the role (P stays for *possible* semantic relation, whereas N for *necessary*).

The results of the test appear quite encouraging since the expected relations between schema elements have been actually found and selected. This has led to a relevant improvement with respect to CtxMatch. In fact, if concepts were simply assembled through the logical operator of conjunction (as CtxMatch does), all the mapping elements would be compatibility relations, $*$. In other words, both versions are correct (we assume that CtxMatch selects the same senses as version 2 for each schema element) with respect to the knowledge available and to the intuitive interpretation of the schemas, but only CtxMatch–2 guarantees the completeness.

The process of word sense disambiguation has reduced the word senses by more than 50%. Both the disambiguation process and the selection of the expected relations are strictly dependent on the background knowledge.

The more *general* the ontologies are, the more semantic relations between the same words we find, because more of their possible meanings are considered

TREE PARSING – LEXICAL ENRICHMENT

k_1

W	music
L	music
SC	*Head*
PS	*Noun*
S	Music#N1 Music#N2 Music#N3 Music#N4 Music#N5

↓

W	classic
L	classic
SC	*Mod*
PS	*Adj*
S	Classic#A1 Classic#A2

k_2

W	baroque
L	baroque
SC	*Head*
PS	*Adj*
S	Baroque#A1

k3

W	organ
L	organ
SC	*Head*
PS	*Noun*
S	Organ#N1 Organ#N2 Organ#N3 Organ#N4 Organ#N5 Organ#N6

LOCAL SEMANTIC ENRICHMENT

Music#N1 → Classic#A1, style#N3 (P)
Music#N3 → Classic#A1, style#N3 (P)

LOCAL SEMANTIC FILTERING

k_1

W	music
L	music
SC	*Head*
PS	*Noun*
S	Music#N1 Music#N3

↓

W	classic
L	classic
SC	*Mod*
PS	*Adj*
S	Classic#A1

k_2

W	baroque
L	baroque
SC	*Head*
PS	*Adj*
S	Baroque#A1

k3

W	organ
L	organ
SC	*Head*
PS	*Noun*
S	Organ#N1 Organ#N2 Organ#N3 Organ#N4 Organ#N5 Organ#N6

Table 8.3: Local processes applied to H_1

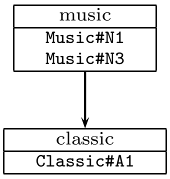
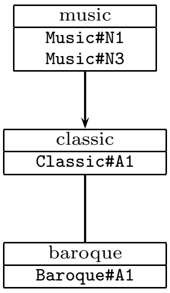
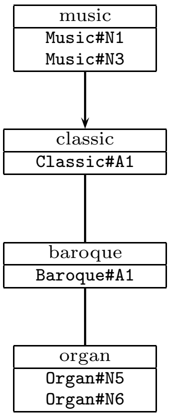
Node	Global enrichment	Global filtering
k_1		
k_2	Music#N1 $\longrightarrow$ Baroque#A1, style#N3 (P) Music#N3 $\longrightarrow$ Baroque#A1, style#N3 (P)	 music Music#N1 Music#N3 classic Classic#A1 baroque Baroque#A1
k_3	Music#N1 $\longrightarrow$ Baroque#A1, style#N3 (P) Music#N3 $\longrightarrow$ Baroque#A1, style#N3 (P) Music#N1 $\longrightarrow$ Organ#N5, play#V3^{-1} (P) Music#N3 $\longrightarrow$ Organ#N5, play#V3^{-1} (P) Music#N1 $\longrightarrow$ Organ#N6 play#V3^{-1} (P) Music#N3 $\longrightarrow$ Organ#N6, play#V3^{-1} (P)	 music Music#N1 Music#N3 classic Classic#A1 baroque Baroque#A1 organ Organ#N5 Organ#N6

Table 8.4: Global processes applied to H_1

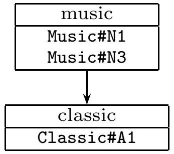
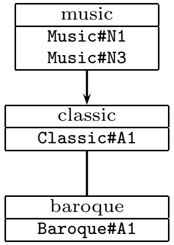
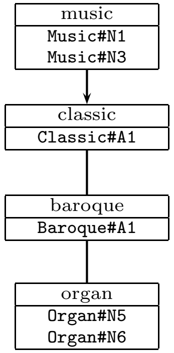
Node	Coverage	Senses induced	Local meaning	Global meaning
k$_1$	Music#N1 $\longrightarrow$ Classic#A1, style#N3 (P) Music#N3 $\longrightarrow$ Classic#A1, style#N3 (P)		k$_1$: (Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Classic#A1	(Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Classic#A1
k$_2$	Music#N1 $\longrightarrow$ Classic#A1, style#N3 (P) Music#N3 $\longrightarrow$ Classic#A1, style#N3 (P) Music#N1 $\longrightarrow$ Baroque#A1, style#N3 (P) Music#N3 $\longrightarrow$ Baroque#A1, style#N3 (P)	 music Music#N1 Music#N3 classic Classic#A1 baroque Baroque#A1	k$_1$: (Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Classic#A1 k$_2$: Baroque#A1	((Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Classic#A1)$\sqcap$ $\exists$style#N3.Baroque#A1
k$_3$	Music#N1 $\longrightarrow$ Classic#A1, style#N3 (P) Music#N3 $\longrightarrow$ Classic#A1, style#N3 (P) Music#N1 $\longrightarrow$ Baroque#A1, style#N3 (P) Music#N3 $\longrightarrow$ Baroque#A1, style#N3 (P) Music#N1 $\longrightarrow$ Organ#N5, play#V3^{-1} (P) Music#N3 $\longrightarrow$ Organ#N5, play#V3^{-1} (P) Music#N1 $\longrightarrow$ Organ#N6, play#V3^{-1} (P) Music#N3 $\longrightarrow$ Organ#N6, play#V3^{-1} (P)	 music Music#N1 Music#N3 classic Classic#A1 baroque Baroque#A1 organ Organ#N5 Organ#N6	k$_1$: (Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Classic#A1 k$_2$: Baroque#A1 k$_3$: Organ#N5 $\sqcup$ Organ#N6	((Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Classic#A1)$\sqcap$ $\exists$style#N3.Baroque#A1$\sqcap$ $\exists$play#V3^{-1}(Organ#N5$\sqcup$ Organ#N6)

Table 8.5: Interpretation of H_1

TREE PARSING – LEXICAL ENRICHMENT

`c`$_1$

W	music
L	music
SC	*Head*
PS	*Noun*

S	`Music#N1`
	`Music#N2`
	`Music#N3`
	`Music#N4`
	`Music#N5`

W	holy
L	holy
SC	*Mod*
PS	*Adj*
S	`Holy#A1`

`c`$_2$

W	Bach
L	Bach
SC	*Head*
PS	*Noun*
S	`Bach#N1` `Bach#N2`

LOCAL SEMANTIC ENRICHMENT

`Music#N1` → `Holy#A1`, `style#N3` (P)

`Music#N3` → `Holy#A1`, `style#N3` (P)

LOCAL SEMANTIC FILTERING

`c`$_1$

W	music
L	music
SC	*Head*
PS	*Noun*
S	`Music#N1` `Music#N3`

W	holy
L	holy
SC	*Mod*
PS	*Adj*
S	`Holy#A1`

`c`$_2$

W	Bach
L	Bach
SC	*Head*
PS	*Noun*
S	`Bach#N1` `Bach#N2`

Table 8.6: Local processes applied to H_2

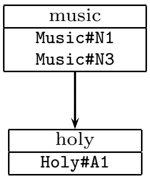
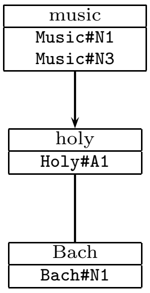
Node	Global enrichment	Global filtering
c_1		
c_2	Music#N1 → Bach#N1, compose#V2^{-1} (P) Music#N3 → Bach#N1, compose#V2^{-1} (P)	 music Music#N1 Music#N3 holy Holy#A1 Bach Bach#N1

Table 8.7: Global processes applied to H_2

and related in the ontology. This increases the complexity of the selection of one coverage as the amount of possible coverages is larger. Furthermore, general ontologies allow to relate a concept to many others and in many different ways. The same concept is in fact not only analyzed and defined in a given small domain but in many different perspectives. The disambiguation process is therefore made very effective because the risk of leaving an element not contextually related (which could cause the selection of all its possible meanings) is very low.

Similarly, the more *precise* the ontologies are, the more concepts are related to others, since for each concept there are many restricting axioms. This increases the complexity of the selection of one coverage as in the case of general ontologies. Moreover, if concept descriptions are very detailed, concepts are unequivocal (intuitively, the mapping on WordNet senses is 1:1 and not 1:n), thus making the disambiguation process very effective. In fact, relations are found between single senses and not groups of senses representing ontology concepts.

We can come to a conclusion: the more general and precise the ontologies, the more complex the coverage choice but the more effective the disambiguation. This empirical though inductive result, given by our methodology, matches intuition: the more our knowledge, the more the possible ways of interpreting schemas but also the more precise the final result.

Finally, notice that most of the ways to relate concepts (all in this case) are semantic relations of the type *possible* (P).

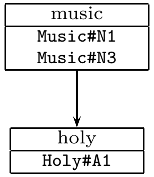
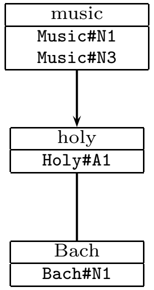
Node	Coverage	Senses induced	Local meaning	Global meaning
c$_1$	Music#N1 $\longrightarrow$ Holy#A1, style#N3 (P) Music#N3 $\longrightarrow$ Holy#A1, style#N3 (P)		c$_1$: (Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Holy#A1	(Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Holy#A1
c$_2$	Music#N1 $\longrightarrow$ Holy#A1, style#N3 (P) Music#N3 $\longrightarrow$ Holy#A1, style#N3 (P) Music#N1 $\longrightarrow$ Bach#N1, compose#V2^{-1} (P) Music#N3 $\longrightarrow$ Bach#N1, compose#V2^{-1} (P)	 music Music#N1 Music#N3 holy Holy#A1 Bach Bach#N1	c$_1$: (Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Holy#A1 c$_2$: Bach#N1	((Music#N1 $\sqcup$ Music#N3)$\sqcap$ $\exists$style#N3.Holy#A1)$\sqcap$ $\exists$compose#V2^{-1}.Bach#N1

Table 8.8: Interpretation of H_2

H_1	H_2	Mapping element
$\mathtt{k}_1$	$\mathtt{c}_1$	$*$
$\mathtt{k}_1$	$\mathtt{c}_2$	$\supseteq$
$\mathtt{k}_2$	$\mathtt{c}_1$	$*$
$\mathtt{k}_2$	$\mathtt{c}_2$	$\supseteq$
$\mathtt{k}_3$	$\mathtt{c}_1$	$*$
$\mathtt{k}_3$	$\mathtt{c}_2$	$\supseteq$

Table 8.9: Mapping between H_1 and H_2

Chapter 9

Related works

Our approach to schema matching is based on the ability to establish the meaning of each schema element according to the information available. This problem requires both the interpretation of strings in natural language (or general expressions commonly used) and the capacity to generate the semantics which is not present in schemas. Both these aspects are subject to scientific research. In this section, we present the main contributions mentioned in literature to the corresponding fields.

We can compare the problem of enriching hierarchical classification with semantic information to the work done by Woods [45] on conceptual indexing. Woods parses each phrase into one or more conceptual structures representing how the elements of the phrase are assembled to form its meaning. Then, he uses taxonomies of concepts identifying generality relationships among the individual elements of the phrase in order to determine when the meaning of a phrase is more general than the meaning of another. For example, he can automatically determine that 'car washing' is a kind of 'automobile cleaning', given the information that a 'car' is a kind of 'automobile' and that 'washing' is a kind of 'cleaning'. Even if similar methodologies are applied, our approach aims at interpreting already existing taxonomies in order to make a number of semantic relations explicit, while in conceptual indexing the starting point is extracting terminology from documents.

The contextual interpretation of schema elements has been also suggested by Kavalec and Svatek [31] with particular emphasis on Web directories. However, their goal is different since they use the knowledge embedded in the structure to obtain labeled training data with limited human effort for information extraction from Web documents. Otherwise, our goal is to interpret schema elements in order to discover the content of the documents classified without analysing them.

The interpretation of node labels and attributes in an ACH requires the identification of the concept to which each word refers in that particular context. This problem is commonly named as *word sense disambiguation*. In literature, it is widely studied as shown in the detailed survey provided by Ide and Veronis [30]. Word sense disambiguation is usually tackled through two macro–steps:

(i) the identification of the word senses relevant to the context (e.g. phrase, text, structure) under consideration and (ii) the assignment of the word to the appropriate sense. The first step usually occurs by associating the word to some specific information which can characterize it, e.g.:

- the corresponding definition (e.g. dictionaries);
- groups of features, categories, or associated words (e.g. synonyms, as in a thesaurus);
- entries in a transfer dictionary (e.g. the corresponding translations in other languages).

However, the precise definition of a sense is still subject to considerable debate within the community and it seems to have not easy and definitive solution in short time (see the argument by Ide and Veronis [30]).

The second step is achieved by relying on two major sources of information:

- the context of the word, in the broad sense;
- external knowledge resources (e.g. lexical and encyclopedic resources), which provide data useful to associate words with senses.

The disambiguation process works like this: it can either match the context where we find the instance of the word to be disambiguated with information from an external knowledge resource (see the approach by Lesk in the following), or it matches the word context with information about the context (derived from corpora) of previously disambiguated instances. Association methods are then used to determine the best match between the current context and one of these information sources. The best match indicates the most likely sense to be associated to each word occurrence.

One of the first works on word sense disambiguation was carried out by Lesk [32]. He proposed to disambiguate the sense of the words in a given context according to the co–occurence of words in the corresponding definitions. For example, given a context as "there was ash from the coal fire", 'ash' was resolved as "the soft grey powder that remains after something has been burnt" instead of "a forest tree common in Britain", since it contained the verb 'to burn', which was also contained in the definitions of coal (i.e. "a black mineral which is dug from the earth, which can be burnt to give heat") and fire (i.e. "the condition of burning; flames, light and great heat"). An evolution of this methodology has been proposed by Manabu in [36].

Another relevant proposal is the one of Agirre and Rigau [18]. They propose a procedure for lexical ambiguity resolution based on an elaboration of the conceptual distance among concepts, i.e. conceptual density [17]. The system clusters words in semantic classes according to the word lexicon and analyzes how classes are hierarchically organised. Then, the system resolves the lexical ambiguity by finding the combination of senses which maximizes the total conceptual density.

This procedure and many others in the field of word sense disambiguation are based on the notion of conceptual distance among concepts. In literature, we can find many different approaches to the reckoning of such a distance. The one proposed by Agirre and Rigau [18] and the one by Resnik [41] are among the most relevant. Resnik states that the more information two concepts share in common, the most similar they are. The information shared by two concepts is indicated by the information content of the concepts that subsume them in a given taxonomy. The information content of a concept is computed as negative the log likelihood, according to the *information theory*. So, the more abstract a concept, the lower its information content.

Many other approaches to word sense disambiguation can be found in literature as the one of Yarowsky [46], of Li, Szpakowicz et al. [33], of Okumura and Honda [38], of Sanderson [42], etc.

Some of these methodologies and our work, too, are based on the use of WordNet [16]. WordNet is an online lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets (e.g. hypernym, hyponym, meronym, holomyn). However, such relations are only few among the possible relations existing between concepts and have a linguistic value mainly. Thus, the use of WordNet as ontological resource is quite limited. Anyway, some researchers are currently involved both in the analysis of WordNet as semantic resource and in its possible extension as ontology. For example, Gangemi, Guarino et al. [26] discuss the semantic limitations of WordNet and propose an upgrade of its top–level synset taxonomy in order to make it more conceptually rigorous and cognitively transparent.

A different approach to the ontological enhancement of WordNet is to align it with existing available ontologies. Ontologies [27] are formal descriptions of the concepts and relations which can exist for an agent or a community of agents. However, concepts and relations are usually represented as strings in ontologies, thus they are subject to the same problem of a natural language text: the word ambiguity. The solutions to the problem are basically two: (i) applying techniques of semantic explicitation (as the one we have presented in this thesis) to the strings representing ontological concepts and relations or (ii) aligning ontologies with a lexical system as WordNet manually or semi–automatically. The first possible solution has not been implemented yet, whereas the second way has been considered in some recent works, e.g. the mapping of WordNet to the SUMO ontology by Niles [37].

One of the most important topics discussed in this thesis and related to ontologies is the extraction of possible semantic relations from knowledge bases. As far as we know, there is no approach in literature which is directly comparable to ours. However, it is our duty to mention some similar works which focus either on ontology exploration or on the extraction of conceptual relations from texts. With respect to the former case, we only remark the existence of a tool called OntoXpl [8] by Ying Lu of the Concordia University. Ontoxpl retrieves the implicit information in a given ontology and reorganizes it in a way such that users can get a global picture of ontology information and explore the knowledge base efficiently (e.g. discovering all the role restrictions of a given concept, finding out all the instances of a given concept). The second category of works is studied in the field of ontology learning, i.e. methodologies to build up and expand existing ontologies (semi–)automatically. Generally, systems for conceptual relation discovery can search both *unnamed* and *named* relations. Unnamed relations express the fact that two concepts are more or less strongly related, although the exact nature of the relation is unknown. In order to find unnamed relations, the technique of *collocations* is usually employed. Collocations are occurrences of two or more words within well–defined units of information (e.g. a document). Significant collocations are selected according to different possible measures as discussed by Smadja in [44], e.g. Heyer, Luter et al. [28] use a measure quite similar to the log likelihood index (applied to the number of sentences containing one of the two words or both of them). Once significant collocations have been found, the relation strength between two given words is usually computed by means of statistical methods. For example, Maedche and Staab [35] use *association rules* and calculate the relation strength according to the *confidence* and the *support* of the two given words. Relevant relations are then identified through suitable thresholds. Note that words in texts are not concepts, however word sense disambiguation can help recognizing the particular concept which the word in the text refers to, thus allowing to discover relations not between words but concepts.

Named relations, i.e. relations with a well–defined type, are stipulated when a concept occurs in a certain expected grammatical pattern in the text. When concepts are encountered in a pattern, a triple is extracted: two concepts and a relation which is found to hold between them. Consider the example proposed by Byrd and Ravin [24]: if we define a pattern as: 'PERSON, ... of ORGANIZATION', it is matched, for example, by this occurrence in the text: "Today, Gerstner, the CEO of IBM, announced that the company...", to yield the conceptual relation *CEO* between *Gerstner* and *IBM*.

A number of variants to the problem exist and usually differ for the class of pattern they use, e.g.:

- patterns which anchor the positions of both concepts and allow for the discovery of the relation name in the remainder of the pattern;

- patterns which anchor the positions of both concepts within a constant pattern and yield a fixed relation name;

- patterns which anchor the position of one of the concepts and of a fixed relation name and allow for the discovery of the second concept.

For a more detailed discussion, refer to the paper by Byrd and Ravin [24] or by Sintex, Junker et al. [43]. Note that all these methodologies are based on the availability of large corpora to discover new relations from texts, whereas our methodology focuses on the extraction of relations defined in existing ontologies, thus not requiring any data instance. Their purpose is to build ontologies, i.e. our exact opposite: exploiting knowledge present in ontologies.

Chapter 10

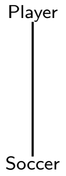
Future works

Our proposal only represents a preliminary version of the CtxMatch–2 algorithm and a lot of work remains to do in order to guarantee pragmatic relevant matchings between concept hierarchies. The efforts must be directed towards the improvement and the refinement of the different steps of the semantic explicitation process. The intended objective is to provide a schema interpretation which matches the classifier's as claimed in Chapter 3. The key of the success of semantic methods resides in the ability of exploiting all the contextual information which is available in the schemata and in the capacity of selecting and filtering the background knowledge which is required for the interpretation of the given schemas.

In the following, we provide a list of possible enhancements of the procedures at the different steps of the semantic explicitation algorithm.

Natural language parsing

The node label parsing is sensible to the improvements of the natural language processing but some more work can be done for tailoring such techniques to structured schemas. Indeed, schemata spread semantically related expressions throughout different nodes, depriving the parsing process of the single node of useful information. The key challange is the adoption of methodologies which are able to guarantee a synergy among the different and independent (in this version) processes of node label parsing. One proposal is about the multiwords recognition across multiple nodes. Consider that the presence of multiwords ease the interpretation process because it minimizes the number of concepts to be semantically linked within a given context. A trivial example is represented by the following phrase, 'soccer player', which is tagged as multiword in the WordNet dictionary. A possible ACH could be rooted at 'player' and it could have a son representing 'soccer' as in the following:



The interpretation of the `Soccer` node would be eased by any process able to recognize that the concatenation of the root node label with 'soccer' results in a multiword. In this case, the `Soccer` node would be associated with the respective multiword concept, Soccer player, and we would be prevented from semantically connecting the two nodes, namely the concept Player and the concept Soccer.

Semantic enrichment and relations extraction

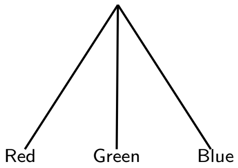
The semantic enrichment is based on the extraction of semantic relations from a knowledge base. This step represents one of the most innovative issues of our methodology but it still requires a lot of study whose results will strongly influence the success of semantic methods. In the Appendix 5, we provide our own definition of semantic relation with respect to a given ontology and an algorithm for deriving connections between concepts. Anyway, other methodologies can be found in literature as the one which proposes of extracting semantic relations from natural language texts according to statistical methods []. The synergy of the different algorithms, their refinement and the filtering of the knowledge on which they work according to the domain of interest represent some of the major challanges in this field.

Some work must be also carried out in the ACH attributes managing. In the current version, we simply connect each possible interpretation of a given node with each possible interpretation of the attribute fillers by means of a DL role representing the disjunction of all the possible intepretations of the attribute meaning. We do not exploit at all the information which is given by the attribute itself. Consider, for example, the `Vacation` node and the respective attribute of Figure 2.1. If we are able of deriving that the only reasonable interpretation of the node is 'vacation (as leisure time) whose location (as point in space) is a country (as geographical region)', we could avoid of linking any other sense of 'vacation' (different from 'leisure time') with any other sense of 'country' (different from 'geographical region'). Moreover, we could disambiguate the attribute role by associating it only with the intended sense. We can modify our algorithm for this purpose with a two–steps procedure: (i) eliminating the steps 12, 18 and 21 in the Algorithm 6.1 and (ii) adding to E_{PT} only those relations which can be derived from the given knowledge and which correspond to one of the possible meanings of the attribute, in the Algorithm 6.2. If we are not able of disambiguating the attribute meaning, we keep all the possible semantic connections as in the current version of the algorithm.

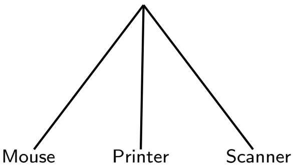
Semantic filtering

The process of semantic filtering is based on heuristic procedures. In the Section 6.4, we presented two rules, respectively the subsumption and the semantic rules, which we apply in the current version of the algorithm. They prefer those senses which are semantically linked to the rest of the context and, in particular, those which are *IsA* or *PartOf*–related. The choice of such criteria was driven by their semplicity and ease of implementation. Furthermore, they represent a point of continuity with respect to the first version of the algorithm [] where their effectiveness has been partially experimented with good results. Anyway, the employment of different procedures represents one of the next challanges. The experimental results will track the way to be followed and will help to find the set and the sequence of heuristics which guarantee the best performances. In particular, the heuristics selectivity will influence the algorithm precision proportionally.

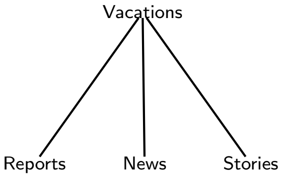
The success of some heuristic procedures also depends on the organization and categorization criteria which have been applied in the schema design. In the following, we present a new proposal which highlights this dependency. In many concept hierarchies and in the most of the taxonomies, we can identify a sort of symmetry or similarity among the siblings of the same node. Consider the following CH examples:



In this case, all the labeled nodes represent a *color*, that is they are *IsA*–related with the Color concept.



'Mouse', 'Printer' and 'Scanner' belong to the same domain. For example, WordNet classifies all of them in the 'computer science' domain.



All the second level nodes can be related to the `Vacation` node by means of the `about` relations.

We can notice that in the first two cases, the nodes have some kind of similarity while in the third case they are semantically related to the same node of the structure by means of the same relation. These measures of similarity can be used for filtering the nodes senses and, consequently, the semantic relations. For example, we could define a rule as follows: *we remove a sense s of a node k if it is not* similar *to a sense of all its siblings and there exists another sense of the same node which has such a property.* Note that we can use many different metrics and clustering methodologies as *similarity* measures: domains as defined in WordNet, ontological distances [1], etc. Analogously, we could define a rule for filtering the nodes senses according to their contextual linkage: *we remove a sense s of a node k if it is not semantically connected to the context in the same way of a sense of all its siblings and there exists another sense of the same node which has such a property.*

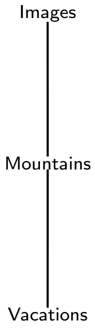
The success of these two heuristics strongly depends on the regularity and symmetry of the schemas to which they are applied. Finally, note that these procedures have the main advantage of exploiting the contextual information in a more integrated way. Indeed, the current version of the algorithm focuses on the path from the ACH root to the node to be interpreted for retrieving the necessary information, while these methodologies involve all the siblings of the path nodes too.

Explicitation of the nodes meaning

The most stimulating challenge for the next period is to study the notion of *horizontal* and *vertical coherence* of the nodes coverages. When we interpret a given node k of an ACH, we choose a coverage, we build the local interpretations of all the path nodes and we compose them by means of semantic relations in order to shape the meaning of k. Note that we build a local interpretation for a specific set of nodes and the contextual meaning only for the node in exam. We could also explicit the contextual meaning of all the path nodes for the sake of efficiency but we would not get any improvement because the contextual meaning of a node changes according to the node we are interpreting. Thus, we cannot exploit the semantic explicitation process of a given node for interpreting its ancestora [2]. Consider the following example:

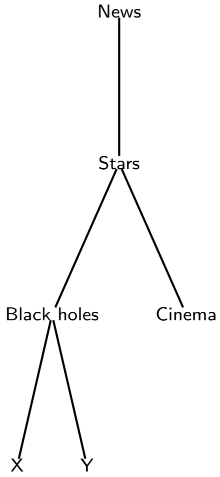
[1] See [], [], ...

[2] This consideration does not hold for taxonomies.



When we interpret the Mountains node, we associate it with the following concept: 'images about mountains', which is very different with respect to the role of the node in the intepretation of the Vacations element. Indeed, the most obvious meaning for the Vacations node is 'images about vacations in mountain'. In the former case, the local concept of Mountains is the modifier of Images by means of the about role, while in the latter case it has become the modifier of the Vacations concept by means of the in role. Anyway, the principles of vertical and horizontal coherence aim at constraining the local interpretation of the single nodes to be constant independently on the schema element we are analyzing. With reference to the above example, we can require that the concept we associate with the Vacations node is 'leisure time' in the interpretation of each node of the structure (which could be much larger). Note that the local interpretations are influenced by the context, thus they cannot be performed independently from it.

We can identify two levels of coherence: (i) *vertical coherence* preserves the meaning of a node along a given path, while (ii) *horizontal coherence* preserves the meaning of a node in different contexts. Consider the following example:



Let us focus on the Stars node. It can refer to 'highly skilled characters' or to 'celestial bodies'. Vertical coherence requires that each descendant agree on its meaning: if we consider its second acception for interpreting the `Blackholes` node, we must keep it for any other node along the same path (i.e. `X` and `Y`) and viceversa. On the other hand, horizontal coherence requires that nodes which share the same context [3] agree on the meaning of the context nodes. Thus, the two nodes `Blackholes` and `Cinema` must be in agreement on the semantics of the `Stars` (and `News`) node. Note that the semantics of a node can be very complex and it can involve more senses of the same word.

The algorithm complexity increases a lot when we apply the principles of coherence. In particular, the COMPOSING–THE–MEANING function needs to check the consistency of the available interpretations of schema nodes each time a new element is processed. When it detects an incoherence, it must identify the (most likely) cause and solve the problem. Our proposal is to have a *backtrack* function which decides, in some sense, which coverage is the most responsible for incoherence. In the following, we propose the COMPOSING–THE–MEANING function of the Algorithm 6.6 with the backtrack enhancement.

```
Algorithm 10.1 COMPOSING–THE–MEANING-B($H_s, k$)
  ▷ semantic ACH: $H_s = \langle H, PT, E_{ACH}, E_{PT}, \mathrm{lab}_s \rangle$
  ▷ node: $k \in H$
  VarDeclarations
      semantic ACH: $H_{s,k} = \langle H', PT', E'_s, lab'_s \rangle$
      set of node coverages: $W$
      coverage: $CK_k$
      set of relations: $E_{k'}$
      parse tree node: $n$
      array of DL terms: $F[\,]$
      set of ACH nodes: $M$
      set of relations: $E_{ACH}$
1   $H_{s,k} \leftarrow$ EXTRACT-SEMANTIC-ACH-GENERATED-BY-NODE($H_s, k$);
2   $W \leftarrow$ EXTRACT-NODE-COVERAGES($H_{s,k}, k$);
3   $CK_k \leftarrow$ CHOOSE-NODE-COVERAGE($W$);
4   UPDATE-AGENDA($k$, $CK_k$);
5   if (AGENDA-IS-CONSISTENT())
6       for each node $k' \in H'$
7           $E_{k'} \leftarrow \{e \in CK_k \mid e = e_{\langle k', \_, \_ \rangle \leftrightarrow}\}$;
8           $n \leftarrow \mathrm{root}(PT'(k'))$;
9           $F(k') =$ BUILD–LOCAL–MEANING($PT'(k'), n, E_{k'}$);
10      $M \leftarrow \{k \in H' \mid \neg\exists e \in CK_k(e = e^{ACH}_{\langle k, \_, \_ \rangle \leftarrow})\}$;
11      $E_{ACH} \leftarrow \{e \in CK_k \mid e = e^{ACH}\}$;
12      $lab_s(k) \leftarrow \bigsqcap_{m_i \in M}$ BUILD–GLOBAL–MEANING($F, m_i, CK_k$);
13  else BACKTRACK();
14  Return $H_s$;
```

Once we have selected the node coverage for the input node k (step 3), we update the agenda containing the choices we have made so far about the local

[3] In the current version, a node context is given by the path from the root to itself.

meaning of the single nodes (step 4). In step 5, we check the coherence of the different interpretations and we apply the same steps of the Algorithm 6.6 if we have detected no conflicts. In case the set of interpretations is not consistent, we apply the backtrack function (step 13). Its tasks can be be summarized as follows:

- Identify the conflict;
- Investigate for the cause;
- Rearrange the interpretations coherently.

The way of performing such steps goes beyond the scope of our preliminary work and represents one of the most stimulating and complex challanges of the next period.

A strictly related consequence of the employment of the principles of coherence and of a backtrack function is that we can completely remove the filtering procedures. Indeed, we leave to the coverages choice step the task of determining the best set of senses and relations which represents the optimum interpretation of a node in the context of its schema. The filtering criteria become the discriminant for selecting the most coherent coverage of a given node with respect to the whole structure.

Chapter 11

Conclusions

The thesis has moved along two distinct but synergetic directions: the investigation of a methodology for schema matching based on semantic methods and its implementation in a prototypal application. The former has allowed us to work out the theoretical foundations of our proposal and to formalize the conditions which guarantee its soundness. The latter has provided for an immediate verification of the effectivity of the approach.

The work has been very complex and has introduced many innovative contributions both from a theoretical point of view and in terms of algorithms and heuristics. Its relevance is both in the field of semantic interpretation of (semi–)structured data and in the schema matching.

The first step has been the presentation and the formal definition of the schemas we have considered: hierarchical classifications with attributes. They are one of the most general and widely spread schemas for the classification of objects and documents. Furthermore, the data model they represent can be adapted to several other models, thus making our dissertation quite general. CtxMatch considers only hierarchical classifications, whereas we have taken into account the possibility to characterize schema content with attributes, too.

The semantics of mappings between hierarchical classifications has been defined in terms of set–theoretical relations between the content of their elements. Furthermore, since we lacked a uniform and general criterium to evaluate the soundness of a mapping, we have proposed one possible definition of soundness which can be assumed as a guideline when evaluating matching methods of classification schemas.

We have then formalized semantic methods as a solution to the matching problem and we have investigated the conditions which guarantee their soundness. The two most relevant conditions can be summarized as follows: (i) we need to produce an interpretation of the schema elements equal to the one by the classifier and (ii) we need the availability of a background knowledge consistent with that of the classifier's. As the second condition lends only to practical considerations, we focused our efforts on the development of an algorithm able to generate reasonable interpretations of schema elements.

We have thus analyzed how the meaning of schema elements can be effectively made explicit. NLP techniques are needed to parse the grammatical structure of the phrases in schemas, whereas lexical knowledge provides the possible meanings of each word in these phrases. Our contribution to this phase is quite limited since existing functions and external resources well support its execution. The most challanging step is to build the semantics which schemas do not express, i.e. the concept which each word actually refers to (word sense disambiguation) and the way concepts are related. Since we think that the sense of a word is likely to be the one which best fits in the word context, the two problems are all dependent on the ability to approximate the semantic relations holding between pairs of concepts.

We have largely investigated this issue resulting in a methodology which is supported by theoretical considerations about the semantics of existential restrictions in DL knowledge bases. Furthermore, we have provided an algorithm for the extraction of semantic relations between pairs of concepts from existing ontologies and we have discussed the conditions which guarantee its decidability.

We have finally presented the algorithms that we need to apply in order to build the meaning of schema elements. In some cases, our contribution has been only in terms of procedures, whereas in other cases we have proposed heuristics to deal with phases solved by humans through sensibility and experience. In fact, the possible interpretation of an element is not unambiguously determined but each human provides its own.

Once we have worked out our methodology, we have developed a prototypal application whose architecture has been discussed in Chapter 7. In particular, we have pointed out its interactions with external libraries and resources and the need of usage of numerous technologies for its realization. The current implementation represents a good starting point to test the effectivity of our methodology. The results of our experiments, though performed on small examples, have shown a relevant improvement with respect to CtxMatch.

The main limitations of our proposal can been seen (i) in the lack of rich ontologies able to support the process of semantic explicitation and comparison and (ii) in the lack of alignment of the concepts in ontologies with WordNet (or any other similar resource) senses. These points represent general problems in the Semantic Web, thus we expect that they will be gradually addressed and solved. For these reasons, it is likely that our methodology cannot succeed in large–scale contexts (e.g. web directories) in the short term. Instead, its most imminent possible application is in specific and restricted domains where a detailed definition of the common background and of the shared terminology is already a current achievement as well as their formalization, in many cases.

This thesis has fully explained the methodology which has been used. However, a lot of work can still be performed in order to enhance algorithms and heuristics as we briefly discuss in the next lines.

The label parsing is subject to the improvements of NLP but some work can be done to tailor such techniques to schemas. In fact, schemas spread semantically related expressions throughout different nodes, depriving the parsing process of the single node of relevant information. The challange is the adoption of techniques which are able to guarantee a synergy among such processes. A proposal may be the recognition of multi–words dispersed across multiple nodes.

Some work must also be carried out in the way attributes are managed. In the current version we do not worry about disambiguating attribute names. However, the node to which they are attached and their filler provide a good context to derive their intended meaning. For example, consider the node `Vacations` of Figure 2.1. If we are given that only the sense 'a determination of the location of something' of 'location' is semantically related to a sense of 'vacation' and of 'country', we may use this information to disambiguate the meaning of the three words.

The extension of the definition of *context* of a node can introduce new heuristics for filtering word senses. In many classification schemas, it exists a sort of symmetry (or similarity) among siblings. For example, all the children of a certain node could be intended as colors or as belongings of the same domain (e.g. it exists a sense of 'mouse', 'printer' and 'scanner' in the domain 'computer science'). Thus, the possible senses of sibling elements can be clustered according to some similarity measure (e.g. domains as defined in WordNet, ontological distance: all techniques already used in word sense disambiguation), which is then used to filter out non–matching senses.

In the next future, we aim at enhancing the methodology and the algorithms according to these directions and then developing a full–featured version of the system. This will result in an application to be used in extensive testing and in performance comparison with the other most relevant methodologies of schema matching. Finally, the architecture will be re–engineered in order to be integrated into the commercial tool KEEx[1], where CtxMatch already runs.

[1] A peer–to–peer tool for distributed knowledge management [22].

Bibliography

[1] DAML Ontology Library. Available at: *www.daml.org/ontologies.*

[2] The DARPA Agent Markup Language. Available at: *www.daml.org.*

[3] The ECl@ss Standardized Material and Service Classification. Available at: *www.eclass-online.com.*

[4] The Kompass B2B search engine. Available at: *www.kompass.com.*

[5] Mikrokosmos' ontology. Available at: *crl.nmsu.edu/mikro.*

[6] Minipar. Available at: *www.cs.ualberta.ca/lindek/minipar.htm.*

[7] The North American Industry Classification System (NAICS). Available at: *www.naics.com.*

[8] Ontology explorer tool (ontoxpl). Available at: *www.cs.concordia.ca/ying_lu.*

[9] OpenCyc. Available at: *www.opencyc.org.*

[10] Pellet OWL Reasoner. Available at: *www.mindswap.org/2003/ pellet/index.shtml.*

[11] Racer. Available at: *www.sts.tu-harburg.de/∼r.f.moeller/racer.*

[12] Saxon: the xslt and xquery processor. Available at: *saxon. sourceforge.net.*

[13] The Suggested Upper Merged Ontology (SUMO). Available at: *www.ontologyportal.org.*

[14] The United Nations Standard Products and Services Code (UNSPSC). Available at: *www.unspsc.org.*

[15] Web Ontology Language (OWL). Available at: *www.w3.org/ 2004/OWL.*

[16] Wordnet: a lexical database for the english language. Available at: *www.cogsci.princeton.edu/wn.*

[17] E. Agirre and G. Rigau. A proposal for word sense disambiguation using conceptual distance. In *Proceedings of the 1st International Conference on Recent Advances in NLP*, Tzigov Chark, Bulgaria, September 1995.

[18] E. Agirre and G. Rigau. Word sense disambiguation using conceptual density. In *Proceedings of COLING-96*, pages 16–22, Copenhagen, Danmark, 1996.

[19] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *Description logic handbook.* Cambridge University Press, 2003.

[20] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[21] M. Benerecetti, P. Bouquet, and S. Zanobini. Soundness of semantic methods for schema matching. In *Proceedings of Meaning Coordination and Negotiation Workshop, in conjunction with the third International Semantic Web Conference (ISWC-04)*, pages 7–11, Hiroshima, Japan, November 2004.

[22] M. Bonifacio, P. Bouquet, P. Busetta, A. Danieli, A. Don, Gianluca Mameli, and M. Nori. KEEx: A Peer-to-Peer Tool for Distributed Knowledge Management. Presented at the 4th International Conference on Knowledge Management (I-KNOW '04).

[23] P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: a new approach and an application. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Second International Semantic Web Conference (ISWC-03)*, Lecture Notes in Computer Science (LNCS), pages 130–145, Sanibel Island (Florida, USA), October 2003. Springer Verlag.

[24] R. Byrd and Y. Ravin. Identifying and extracting relations in text. In *Proceedings of NLDB*, Klagenfurt, Austria, 1999.

[25] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of WWW-2002, 11th International WWW Conference, Hawaii*, 2002.

[26] A. Gangemi, N. Guarino, and A. Oltramari. Conceptual analysis of lexical taxonomies: the case of wordnet top-level. In *Proceedings of the international conference on Formal Ontology in Information Systems*, pages 285–296. ACM Press, 2001.

[27] T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[28] G. Heyer, M. Luter, and U. Quasthoff. Learning relations using collocations. In *Proceedings of the IJCAI Workshop on Ontology Learning*, Seattle, USA, August 2001.

[29] I. Horrocks and U. Sattler. Ontology reasoning in the SHOQ(D) description logic. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[30] N. Ide and Jean Veronis. Introduction to the special issue on word sense disambiguation: the state of the art. *Comput. Linguist.*, 24(1):2–40, 1998.

[31] M. Kavalec and V. Svatek. Information extraction and ontology learning guided by web directory. In *ECAI Workshop on NLP and ML for ontology engineering*, Lyon, 2002.

[32] M. Lesk. They said true things, but called them by wrong names – vocabulary problems in retrieval systems. In *Proceedings of the 4th Annual Conference of the University of Waterloo Centre for the New OED*, 1988.

[33] X. Li, S. Szpakowicz, and S. Matwin. A wordnet-based algorithm for word sense disambiguation. In *IJCAI*, pages 1368–1374, 1995.

[34] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *The VLDB Journal*, pages 49–58, 2001.

[35] A. Maedche and S. Staab. Discovering conceptual relations from text. In *Proceedings of the 14th European Conference on Artificial Intelligence, IOS Press*, pages 321–325, Amsterdam, 2000.

[36] O. Manabu. Word sense disambiguation by marker passing on very large semantic networks. In *Proceedings of the Natural Language Processing Pacific Rim Symposium*, pages 71–76, 1995.

[37] I. Niles and A. Pease. Linking lexicons and ontologies: Mapping wordnet to the suggested upper merged ontology. In *Proceedings of the 2003 International Conference on Information and Knowledge Engineering (IKE 03)*, Las Vegas, Nevada, June 2003.

[38] M. Okumura and T. Honda. Word sense disambiguation and text segmentation based on lexical cohesion. In *Proceedings of the Fifteen Conference on Computational Linguistics (COLING-94)*, volume 2, pages 755–761, 1994.

[39] H. Putnam. *Reason, Truth, and History.* CUP, 1981.

[40] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 2001.

[41] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, 1995.

[42] M. Sanderson. Word sense disambiguation and information retrieval. In *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*, pages 49–57, Dublin, IE, 1994.

[43] M. Sintek, M. Junker, L. van Elst, and A. Abecker. Using information extraction rules for extending domain ontologies. In *IJCAI'2001 Workshop on Ontology Learning, Proceedings of the Second Workshop on Ontology Learning OL'2001 in conj. with the 17th International Conference on Artificial Intelligence IJCAI'2001*, Seattle, USA, August 2001.

[44] F. Smadja. Retrieving collocations from text: Xtract. *Comput. Linguist.*, 19(1):143–177, 1993.

[45] W. Woods. Conceptual indexing: A better way to organize knowledge. Technical Report TR-97-61, Sun Microsystems Laboratories, April 1997.

[46] D. Yarowsky. Word-sense disambiguation using statistical models of Roget's categories trained on large corpora. In *Proceedings of COLING-92*, pages 454–460, Nantes, France, July 1992.