Learning data-consistent mappings from a relational database to an ontology

Benjamin Habegger

Dipartimento di Informatica e Sistemistica
Università di Roma 1 "La Sapienza"
Via Salaria 113, 00198 Rome, Italy
Email: habegger@dis.uniroma1.it

Abstract. Determining how to map information between different representational models is an essential task of semantic integration. Up to now, most research in this field has concentrated on the problem of finding pairs of elements which could have similar semantics (ie. finding *matches*). From the work we have seen, it is not clear that any guarantees can be given that the obtained matches respect the intended semantics of the target model. We also argue that *matches* are not sufficient to clearly define *mappings* which respect the intended interpretation of the target model. In this paper, we define mapping *consistency* with respect to an intended interpretation and derive a notion of *data-consistency*, a necessary condition for consistency. From this, we propose a relational learning algorithm to construct data-consistent *mappings*.

1 Introduction

Building mappings between schema and/or ontologies is one of the central issues of semantic integration [1]. Hand crafting mappings is a complex, tedious and error-prone task. Much research on how to (partially) automate semantic integration has been lead in both the database [2, 3] and ontology [4] communities. Most of this work has concentrated on finding correspondences, also called *matches*, between schema/ontology elements using some form of heuristics.

There are two major drawbacks to existing approaches. Firstly, the approaches we have knowledge of do not rely on any formal notion of intended interpretation. Most implicitly rely on the quite arguable intuition that two separately conceived models of similar "worlds" (eg. a computer science department) will contain similar concepts/roles with similar characteristics. They therefore can give no guarantee on the correctness (with respect to the intended interpretation) of the obtained matchings. Secondly, having matches is far from being sufficient in order to correctly be able to map data between representations. Correct reasoning (and in particular querying) over integrated ontologies and/or databases requires having mappings which respect the intended interpretation of the integration system. In formal frameworks which have been proposed for both database and ontology integration [5, 1], mappings usually take the form of complex queries. The few systems which do allow to build mappings [6], require

extra knowledge (eg. foreign keys) and only produce mapping suggestions with no guarantee of their correctness.

In this paper, we propose a framework defining mapping consistency with respect to an intended interpretation (ie. the one given by an expert) and derive a notion of *data-consistency* allowing to check if a mapping is compatible with a previously given set of positive and negative examples. Given that the examples are in accordance with the intended interpretation, data-consistency is a necessary condition for consistency. From this we adapt existing inductive logic programming techniques to build data-consistent mappings. We consider here the problem of mapping a relational database into an ontology in a global as view setting. There are mainly two aspects which make mapping databases (rather than ontologies) a particular problem. First, databases usually have very few (if any) intentional knowledge which can be exploited to build mappings. Second, the objects which are described by a relational database do not have an explicit representation as they do in ontologies. The mappings we consider consist of a set of horn clauses where the head of each clause is a concept or role of the target ontology and the body is a conjunctive query over the relations of the source schema. In the work presented here, we will consider the case where each concept or role of the target schema can be described by a unique rule.

The contributions of this paper are the following : (1) We introduce the notion of mapping *consistency* with respect to a given intended interpretation. (2) We show how a reduced form of consistency, data-consistency, can be tested in the presence of examples. (3) We propose a relational learning algorithm allowing to build data-consistent mappings.

This paper is organized as follows. Section 2 presents related work in semantic integration an positions our work. Section 3 presents how we handle instance-level mappings. Section 4 introduces our working example. Mapping consistency and data-consistency are presented in section 5. Section 6 proposes a framework which allows to learn data-consistent mappings. Finally, in section 7 we conclude and discuss future work.

2 Background and related work

Building mappings from one model to another requires to overcome multiple difficulties both at the conceptual and instance levels. At the conceptual level choices such as which information should be made explicit, how detailed the information should be or how it should be organized may differ. For example, one model might explicitly define a $CSProfessor$ concept while in another model this information might be left implicit (as a professor who teaches a computer science course). At the instance level, different levels of detail may exist between attribute values. For example, one source may split into pieces an "address" field which another source may represent as a whole. Another important aspect at the instance level is that sharing instances between sources requires having a common naming. The major difficulty to overcome is that any instance (eg. a

particular person) should be represented by a unique object identifier in a target ontology while it is represented as a particular instance of a table in a database.

Integrating data from one source to another has two major requirements : (1) knowing how instances are represented (identified) in each of the models and how to map from one representation to the other and (2) knowing how to translate relationships which hold between instances. In both cases, mappings may be used to represent the relationships. The problems of *instance mapping* and *schema mapping* respectively seek in resolving these two requirements.

Defining mappings between sources is a tedious and error-prone task. Researchers have therefore looked to (partially) automate semantic integration. Up to now most research, whether in the database [2] or the ontology [4] communities, has concentrated on methods which semi-automatically search for correspondences between schema elements (eg. attributes in the relational case). A correspondence between a set of elements A of one schema to a set of elements B of another schema is called a *match* (noted $A \cong B$). When a set of elements is greater than one, one must also define how to combine the values. A particular match might put into correspondence an attribute *name* of a relation $Person_S$ of a source schema with the pair of attributes $firstName$ and $lastName$ of a $Member_T$ relation of a target schema using the concatenation function to join the first and last names.

Different approaches to finding matches have been proposed. Many systems (eg. [7–10]) search for pairs of "similar" elements according to different heuristics. Typical heuristics are of the form, "concepts with similar names are likely to be similar". The heuristics are usually formalized as similarity measures which may take into account different types of information including for example the names and textual descriptions (eg. comments in the schema) of the elements, constraints on the values the elements can take (eg. primary key, type, uniqueness). [11] give a list of the most commonly used informations. Most approaches also consider the similarity of related elements. [7] base their similarity measures on the basis of a distance to elements known *a priori* to be the same.

Other approaches such as the LSD [12] and GLUE [13] systems make use of machine learning over instance data to build matches. LSD learns classifiers based on previously hand-coded matches to predict how to match unseen schemas and combines multiple classifiers some of which rely on having instance data. GLUE learns to predict similarity between two elements of two sources by estimating the joint probability that an instance belongs to both elements. Until recently most systems were only able to suggest one-to-one matches between elements. More recent systems such as iMAP allow to suggest complex matches (eg. $price = rate * (1 + taxes)$) [14]. The suggestions are obtained by multiple predefined searchers such as text searchers looking for concatenations.

While matches can be helpful to integrate data, we argue that they are not sufficient to define mappings, since a set of matches can be interpreted in several ways and some information may be missing. For example, knowing that $Professor_T$ *matches* $Member_S$ does not allow to determine a *mapping* such as

$$Professor_T(x) \leftarrow Member_S(x), Profession(x, "professor")$$

In the literature there have only been few attempts to (partially) automate *mapping* construction. An example, is the Clio system [6], which suggests *mappings* given a set of *matches*. In particular it searches for joins between relations by following foreign keys. Depending on existing matches and the possible presence of constraints can be restrictive. This work does provide a notion of correctness with respect to the constraints of the source and target schemas but not to the *intended interpretation*. The approach proposed in this paper suggests *learning* by examples the constraints which must be satisfied by the data to correctly reflect the intended meaning the mappings should expose.

In this paper, we consider using machine learning to build the mappings of an integration system between a source database and a target ontology in a global as view perspective. Following [5], we are building an integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where $\mathcal{G}$ (resp. $\mathcal{S}$) is the global ontology (resp. source schema) expressed in a language $\mathcal{L}_\mathcal{G}$ (resp. $\mathcal{L}_\mathcal{S}$) over an alphabet $\Sigma_\mathcal{G}$ (resp. $\Sigma_\mathcal{G}$) and $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$. In our framework $\mathcal{M}$ is a set of horn clauses $p_\mathcal{G}(\overrightarrow{x}) \leftarrow conj_\mathcal{S}(\overrightarrow{x}, \overrightarrow{x})$ where $p_\mathcal{G} \in \Sigma_\mathcal{G}$ and $conj_\mathcal{S}(\overrightarrow{x}, \overrightarrow{x})$ is a conjunctive query over $\Sigma_\mathcal{S}$. When learning will we consider a particular interpretation domain Δ and a given database instance $\mathcal{DB}$ of $\mathcal{S}$ over Δ as well as a set of positive and negative instantiation assertions (resp. $\mathcal{KB}^+$ and $\mathcal{KB}^-$) over Σ_S and Δ. As usual, the semantics of an ontology (similarly for a database) $\mathcal{O}$ defined over an alphabet $\Sigma_\mathcal{O}$ is given by an interpretation $\mathcal{I} = (\Delta, \cdot^\mathcal{I})$ where Δ is the domain of interpretation and $\cdot^\mathcal{I}$ is an interpretation function which associates each concept C (resp. each role R) in Σ_O a subset of elements (resp. couples) of Δ.

The objects of the interpretation domain Δ will have been obtained after an initial identification of unique object identifiers as described in section 3. Our framework currently only considers that the source is a flat database and therefore contains only positive assertions and no intentional knowledge. Also, we currently require learning independent mappings for each concept and role of the target ontology.

3 Mapping instances to objects

	Telephone		
	Fname	*Lname*	*Telephone*
people("Mary","Green")	"Mary"	"Green"	0768
people("Mary","Green")	"Mary"	"Green"	0679
people("John","White")	"John"	"White"	0687

Fig. 1. An example relation and the object identifiers of each tuple

It is common to hypothesize that both the target and source databases are defined over a common fixed interpretation domain Δ. In the following we will

be proceeding similarly. However, this requires having a common object space, ie knowing how to map the representation of objects in the concrete source database (eg. the instance of the table *members* whose attribute *id* have the value 165) into a unique representation at the conceptual level of the target ontology (eg. the person 'john'). In this section we describe how such identifiers can be obtained.

For any given database, we can consider that each line of a table contained in the database corresponds to an object. In the most favorable case, primary and foreign keys have be defined in order to determine relationships between tables. This information can be used to define object instances and relationships between them. In some cases no primary keys have been given and connections between tables have not been made explicit or the keys do not give the correct view of the data and therefore we will not rely on having such data.

We will rather simply, consider that the user provides for each relation R a set of attributes $ident(R)$ allowing to construct an identifier for each instance which corresponds to the target view he wishes to have of the data. This identifier will be considered to refer to a unique object. Consider a relation R of a given database $\mathcal{DB}$ and a set of identification attributes $K = ident(R)$ for R. Any instance of R is a tuple $(\overrightarrow{x})$ which can be uniquely identified in R by $\overrightarrow{y} = \pi^R_K(\overrightarrow{x})$, where $\pi^R_K(\overrightarrow{x})$ represents the selection of values in $\overrightarrow{x}$ corresponding to the attributes K in R. If we have a set of identification attributes for each relation and associate to each relation a unique functor f_R, any object described by the database can be assigned a unique identifier by applying this functor to the values of the identification attributes. We can complete the domain of interpretation of the original database Δ_v (v for values) with a domain of interpretation for objects

$$\Delta_o = \bigcup_{R \in \Sigma_\mathcal{S}} \left\{ f_R(\pi^R_K(\overrightarrow{x}))/(\overrightarrow{x}) \in R, K = ident(R) \right\}$$

Figure 1 gives an example relation where the attributes $FName$ and $LName$ have been specified as the identification attributes. The relation also includes a specific object identifier where the values correspond to the unique object identifiers for the table. As the figure shows, an identifier is not necessarily a primary key for the table. For example, in the figure both "Mary Green" instances are considered to refer to the same person and which in this case would have two telephone numbers.

In the remainder of this work we will consider that identifiers have been previously determined and added explicitly to the relations of the database and the domain of interpretation shared between the target ontology and source database will be the set $\Delta = \Delta_o \cup \Delta_v$. For readability, we will use short names for each object. For example, we will use $john$ as an object identifier instead of a functor notation such as $people("John","White")$.

4 Introductory example

Let us consider the following example situation. We have a university with different departments, one of which is the computer science department. The uni-

versity wishes to make use of the data which is already made available in existing department databases. However, it wants to have access to this data given its *own predefined ontology*. This requires building mappings between the department schema's and its own ontology. Our working example will consider building the mappings from the CS department schema to the university ontology.

$People_d$		
Oid	$Ident$	$Name$
mary	1	"Mary"
john	2	"John"
charlotte	3	"Charlotte"
fred	4	"Fred"

$Teams_d$			
Oid	Id	$Name$	$Director$
dbteam	1	"Databases"	2
tcteam	2	"Theorectical CS"	4
mlteam	3	"Machine Learning"	1
systeam	4	"Systems"	3

$Teaches_d$	
$Teacher$	$Course$
2	1
2	3
1	2
5	4
4	5

$Courses_d$			
Oid	Id	$Name$	$CSCourse$
db	1	"Databases"	true
ilp	2	"ILP"	true
stats	3	"Statistics"	false
algo	4	"Algorithmics"	true
algebra	5	"Algebra"	false

(a) Computer science database

$Professor_u^+$
john
mary
fred

$Professor_u^-$
jules
charlotte

$CSProfessor_u^+$
john
mary

$CSProfessor_u^-$
jules
charlotte
fred

(b) Target concepts

$$Professor_u(X) \leftarrow People_d(X, Y, \_), Teams_d(\_, \_, \_, Y), Teaches_s(Y, \_).$$
$$CSProfessor_u(X) \leftarrow People_d(X, Y, \_), Teams_d(\_, \_, \_, Y), Teaches_d(X, Y), Courses_d(\_, Y, \_, "true").$$

(c) Target mappings

Fig. 2. An example mapping learning problem

The CS department schema contains four relations : $People_d$ which describes the members of the department, $Teams_d$ which describes the teams of the department and who they are directed by, $Teaches_d$ which relates members to the courses they teach and $Courses_d$ which contains the set of courses members of the department participate in. In this department, all professors both direct a team and teach a (computer science) course. This information is implicit and has not directly been modeled. Figure 2(a) gives an example partial database using the schema of the CS department. The relations have a d index to denote that they belong to the department's schema.

We will consider two scenarios. In the first, the university ontology contains (among others) a $Professor_u$ concept and, in the second, it contains a more specific $CSProfessor_u$ concept. In the first (resp. second) scenario, the intended interpretation of $Professor_u$ (resp. $CSProfessor_u$) is any member of the computer science department which teaches any course (resp. a computer science course) and directs a team. Figure 2(b) gives the set of instances which appear in the partial database of figure 2(a) and should (resp. should not) belong to each of the two target concepts and the correct target mappings for each of them. In the approach we will describe afterwards, these mappings can be learned by generalizing the descriptions of a set of example instances (not necessarily all) we know to belong (or not) to the target concept or role.

5 Intended interpretation

In the context of this paper we only consider the alphabet Σ of symbols associated to the elements of an ontology or schema. In the following, the word schema may therefore often refer to its associated alphabet.

During the conception of a schema or ontology, the conceiver has in mind a specific semantics for the schema elements : their "intended" semantics. In some way, this means that if the conceiver was presented any "world" to be modeled by his schema he would be capable of giving the "intended" semantics of his schema for this world (ie. construct the (extensional) database for which this "world" is a model). Therefore the conceiver or any person or entity having a complete understanding of the schema can be modeled as an "expert oracle".

Definition 1 (Expert oracle). *An expert oracle for a given alphabet Σ can be modeled as a function O_Σ which returns for any domain of interpretation Δ an interpretation function $\cdot^{\mathcal{I}}$*

In the previous definition, the oracle is defined as a function and thus each alphabet yields a unique interpretation for each world. This simply suggests that there is no ambiguity in the idea the conceiver has of the predicates appearing in Σ. Once we have an oracle O_Σ for our schema, the intended interpretation of a given world is the one returned by the oracle.

Definition 2 (Intended interpretation). *Given a schema Σ an expert oracle O_Σ, and a world Δ, the intended interpretation of Σ over Δ according to O_Σ is $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}} = O_\Sigma(\Delta))$.*

As previously stated, we will consider working over a common domain of interpretation Δ. Given a schema Σ what remains to be determined by the intended interpretations is the set of tuples belonging to each relation in Σ. In the following we will denote this set by $\Delta^{\mathcal{I}}$ given the intended interpretation $\mathcal{I}$.

From a general point of view, a mapping from a source schema Σ_S to a target schema Σ_T can be seen as a function m which transforms any database over Σ_S into a database over Σ_T. To simplify discourse, we will consider a database to be a set of ground atoms, which is the case when we have no intentional knowledge.

A mapping m is consistent when the intended interpretation of Σ_T (over Δ) contains the database obtained by applying m to the intended interpretation of Σ_S (over Δ). m is complete when its application to the intended interpretation of Σ_S contains the intended interpretation of Σ_T. Whenever it is both consistent and complete, we say that m is a perfect mapping.

Definition 3 (Mapping consistency and completeness). *Let m be a mapping from a source schema Σ_S to a target schema Σ_T, $\mathcal{I}_S$ and $\mathcal{I}_T$ respectively be the intended interpretations for Σ_S and Σ_T over Δ.*

- *m is* consistent *iff $m(\Delta^{\mathcal{I}_\mathcal{T}}) \subseteq \Delta^{\mathcal{I}_\mathcal{S}}$*
- *m is* complete *iff $\Delta^{\mathcal{I}_\mathcal{S}} \subseteq m(\Delta^{\mathcal{I}_\mathcal{T}})$*
- *m is perfect if it is both complete and consistent (ie. $\Delta^{\mathcal{I}_\mathcal{S}} = m(\Delta^{\mathcal{I}_\mathcal{T}})$)*

When working with real databases, we only have access to a subset of Δ. Furthermore in many cases, the effective database we work with is incomplete in the sense that some relations holding among objects of Δ we know of, might not appear in the database. In some cases it might be inconsistent with the intended interpretation in that some relations which appear in the effective database do not hold in the intended interpretation. In the following work on learning mappings, we will consider that we may have incomplete data but that it is consistent with intended interpretation. If Σ denotes a schema, $\mathcal{DB}$ denotes the effective database over Σ and $\Delta_e \subseteq \Delta$ denotes the subset of objects we have a description for in $\mathcal{DB}$ and $\mathcal{I}$ denotes the intended interpretation of Σ over Δ we have $\mathcal{DB} \subseteq \Delta_e^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.

When considering mapping a source schema $\mathcal{S}$ into a target ontology $\mathcal{T}$ we will consider situations where we have an effective database $\mathcal{DB}_S$ over an effective subset of objects $\Delta_S \subseteq \Delta$. Also we will consider two effective sets of instantiation assertions $\mathcal{KB}_T^+$ and $\mathcal{KB}_T^-$ over an effective subset of objects $\Delta_T \subseteq \Delta$ (usually $\Delta_T \subseteq \Delta_S$ will also hold). $\mathcal{KB}_T^+$ (resp. $\mathcal{KB}_T^-$) is a subset of assertions we know that hold (resp. do not hold) among objects of Δ_T in $\Delta^{\mathcal{I}}$.

In this context and given a mapping m we can define a notion of consistency *with respect to the effective data* we have which we will call *data-consistency*. Of course, data-consistency is a necessary condition for consistency.

Definition 4 (Data-consistency). *Given the effective databases $\mathcal{DB}_S$, $\mathcal{KB}_T^+$ and (eventually) $\mathcal{KB}_T^-$, a mapping m is* data-consistent *with respect to $\mathcal{DB}_S$, $\mathcal{KB}_T^+$ and $\mathcal{KB}_T^-$ iff $\mathcal{KB}_T^+ \subseteq m(\mathcal{DB}_S)$ and $\mathcal{KB}_T^- \cap m(\mathcal{DB}_S) = \emptyset$*

6 Learning mapping rules

We now propose an approach to learning mappings which is data-consistent. As an inductive hypothesis, we will consider that having seen sufficient data will allow us to consider the mapping to hold for remaining data. Therefore, to learn a mapping between a source schema Σ_S and a target schema Σ_T, we will consider having a database $\mathcal{DB}_S$ over Σ_S and two sets of respectively positive and negative instantiation assertions $\mathcal{KB}_T^+$ and $\mathcal{KB}_T^-$.

In our framework, we consider mappings which are defined as a set of horn clauses where the predicate of the head is in the target schema and the predicates of the atoms of the body are in the source schema. A mapping only exists between the source and target languages if the target concept can be described precisely using the source predicates. When building mappings, we will consider this to be true. When learning a target concept C (resp. role R) we will consider the restriction of $\mathcal{KB}_T^+$ (and $\mathcal{KB}_T^-$) to the subset of atoms which belong to C (resp. R). $\mathcal{DB}_S$ will however not be restricted in any manner.

6.1 Instance descriptions

When building a mapping for a target concept C (and similarly for a role R), we are trying to build a mapping rule which with the given database $\mathcal{DB}_S$ entails at least the positive examples and no negative examples (ie. ensures data-consistency). If we consider a particular concept $Professor_u$, we are looking for a description of the instances of the concept $Professor_u$ using the language constructs of the schema Σ_S. In some sense, we hypothesize that the whole database describes that each particular instance belongs to the target concept. For each instance of the target concept, we could build a clause $C(o_1) \leftarrow \mathcal{DB}(o_1)$ where $\mathcal{DB}(o_1)$ denotes that we have a special focus on the instance o_1. The most specific clause which is consistent with a set of positive instance $C^+ = \{o_1, \ldots, o_n\}$, would therefore be the least general generalization of the clauses $C(o_i) \leftarrow \mathcal{DB}(o_i)$. However, such an approach will likely require many positive examples in order to generalize over common content of $\mathcal{DB}$ and still contain sub-clauses which are specific to the database itself and not to the examples.

$$
\begin{aligned}
desc^1_{\mathcal{DB}}(john) =& People_d(john, 2, "John").\\
desc^1_{\mathcal{DB}}(2) =& People_d(john, 2, "John"), Teaches(2,1), Teaches(2,3),\\
& Teams_d(tcteam, 2, "TheoreticalCS", 4),\\
& Teams_d(dbteam, 1, "Databases", 2).\\
desc^2_{\mathcal{DB}}(john) =& desc^1_{\mathcal{DB}}(john), desc^1_{\mathcal{DB}}(2), desc^1_{\mathcal{DB}}("John").
\end{aligned}
$$

Fig. 3. Description of level 1 and 2 of the instance $john$

Given a particular instance of interest, there are some facts (atoms of the database) which have a greater importance depending on there "proximity" to the target instance. Given an instance x, we will denote by $desc^1_{\mathcal{DB}}(x)$ the set of atoms which appear in $\mathcal{DB}$ where x appears as one of the arguments. For example, the first equation of figure 2(a) gives the description of $john$ if the database $\mathcal{DB}$ is the one of figure 2(a).

In order to take into account more distant relationships which could be required to define the target concepts, the description of an instance/value can be extended with the descriptions of the instances/values of the atoms to which it is connected. For example, $john$ is connected to the values 2 and "John" through

the $People_d$ relation. The description of *john* can be extended as show in figure 3. It should be noted that in the presence of foreign keys, we may restrict description extensions to only follow foreign keys.

Definition 5 (Description of an instance). *Given a database $\mathcal{DB}$ and an instance x, the descriptions of level i are defined recursively as follows :*

$$desc^1_{\mathcal{DB}}(x) = \{A(\ldots, x, \ldots) \in \mathcal{DB}\}$$
$$desc^i_{\mathcal{DB}}(x) = \bigcup_{y \in instances(desc^{i-1}_{\mathcal{DB}}(x))} desc^{i-1}(y)$$

The complete description of an instance x is the fix point obtained by gradually incrementing in level, $desc^*_{\mathcal{DB}}(x) = desc^n_{\mathcal{DB}}(x)$ where n is the smallest integer such that $desc^n_{\mathcal{DB}}(x) = desc^{n+1}_{\mathcal{DB}}(x)$. It should be noted that, without any intentional knowledge, a database is always a finite set of atoms. In this case, the complete description of any instance always exists and is finite (at worst it is the whole database itself). Also, given a database $\mathcal{DB}$ where the underlying relational structure which has only one connected component then for any instance x we have $desc^*(x) = \mathcal{DB}$. Otherwise, $desc^*(x)$ is the set of atoms which form the connected component to which x belongs.

6.2 Learning mappings from instance descriptions

In this section we present our mapping learning approach based on the generalization of instance descriptions. The general idea behind our approach is that the description of an instance can be seen as a query which will at least return the instance itself as a result. Generalizing the complete descriptions of the set of instances we know to belong to the target concept will generate a query which will return the initial instances and a subset of the target concept as long as the mapping can be described as a conjunctive query over the source database. However, generalizing the complete descriptions will likely be both infeasible and generate over specific mappings. The approach we propose is to initially learn mappings using only a local context (low description depth) which will be gradually augmented if required. The intuition is that, what is likely to characterize a concept are the features which are the most directly related to the instance.

The generalization operator we will be using is the least general generalization (lgg) defined by Plotkin [15] which is based on the notion of θ-subsumption.

Definition 6 (θ-subsumption). *A clause C θ-subsumes a clause C' (noted $C \succeq C'$) iff there exists a substitution θ such that $C \subseteq C'\theta$.*

Plotkin's least general generalization, given two clauses C and C', produces the most specific (with respect to θ-subsumption) clause $lgg(C, C')$ which generalizes both C and C'.

Definition 7 (Least general generalization). *The* least general generalization *$lgg(C_1, C_2)$ of two clauses C_1 and C_2 is a clause C such that : (1) $C \succeq C_1$, (2) $C \succeq C_2$ and (3) no other clause C' such that $C \succeq C'$, verifies (1) and (2).*

Plotkin has shown that the least general generalizations of two clauses is unique and can be calculated in polynomial time. However, the obtained clause is not necessarily reduced. Reducing a clause requires testing θ-subsumption between clauses which is know to be NP-complete. Recent work have shown that efficient implementations [16] remain tractable in many practical cases. Furthermore, in most mappings will likely not require high depths of descriptions and therefore the clauses to generalize will likely remain small.

Algorithm 1 Algorithm $learn(\mathcal{DB}, E^+, E^-)$

Input A database base $\mathcal{DB}$, positive examples E^+ and negative examples E^-
Output A mappings M which covers all E^+ and no E^-

```
d ← 0; M ← ∅
while M is not consistent with the examples do
  {Expand description depth and restart learning}
  d ← d + 1; M ← ∅
  while M consistent with E⁻ and there is a T(v⃗) ∈ E⁺ do
    C ← description_d(v⃗)
    M ← lgg(E ← C, M)
  end while
end while
return M
```

Algorithm 1 gives describes how a mapping is learned given a set of positive and negative examples. The *lgg* operator is the one given in definition 7. The *description* function calculates the description of an instance as described previously.

The mapping generation process can easily be made interactive. The user starts by given some positive examples from which the system builds a mapping and applies it to the source ontology. This results in a set of tuples which the mapping would transform into the target concept. From these results, the user can either remove instances which have wrongly been added (generating negative examples) and/or add new instances as positive examples.

7 Conclusion and future work

In this paper we have proposed an approach to learning mappings between a database containing the effective data and an ontology. An important distinction with related work is that our approach builds *mappings* which completely define how to transform the representation of instances from one model to another. Currently, most existing efforts to automate semantic integration have concentrated on building *matches* which only relate schema elements together. Furthermore, we have propose a notion of *data-consistency* which allows to have a minimum of guarantee that the semantics of the obtained mappings are correct.

The current work is mostly at its beginning. We have already started an implementation of the ideas developed in this paper and plan to finish them in a near future. We are also willing to test the approach on real-world mapping construction problems. If the tests are successful we plan to extend the framework to handling multiple sources. The current approach does not yet consider existing *matches* which could be made available by using existing semantic integration approaches. We have started studying how these could be used to generate initial mappings which could then be corrected with respect to data-consistency.

References

1. Lenzerini, M.: Data integration: A theoretical perspective. In Popa, L., ed.: PODS, ACM (2002) 233–246
2. Doan, A., Halevy, A.Y.: Semantic Integration Research in the Database Community : A Brief Survey. AI Magazine (2005)
3. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10** (2001) 334–350
4. Noy, N.F.: Semantic integration: A survey of ontology-based approaches. SIGMOD Record **33** (2004) 65–70
5. Calvanese, D., Giacomo, G.D., Lenzerini, M.: Ontology of integration and integration of ontologies. In Goble, C.A., McGuinness, D.L., Möller, R., Patel-Schneider, P.F., eds.: Description Logics. Volume 49 of CEUR Workshop Proceedings., CEUR-WS.org (2001)
6. Popa, L., Velegrakis, Y., Miller, R.J., Hernández, M.A., Fagin, R.: Translating web data. In: VLDB. (2002) 598–609
7. Noy, N.F., Musen, M.A.: The PROMPT suite: interactive tools for ontology merging and mapping. International Journal Human-Computer Studies (2003)
8. Euzenat, J., Valtchev, P.: Similarity-based ontology alignment in owl-lite. In de Mántaras, R.L., Saitta, L., eds.: ECAI, IOS Press (2004) 333–337
9. Palopoli, L., Terracina, G., Ursino, D.: Dike: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. Softw., Pract. Exper. **33** (2003) 847–884
10. Castano, S., Antonellis, V.D., Melchiori, M.: Artemis: A process modeling and analysis tool environment. In Ling, T.W., Ram, S., Lee, M.L., eds.: ER. Volume 1507 of Lecture Notes in Computer Science., Springer (1998) 168–182
11. Ehrig, M., Sure, Y.: Ontology mapping - an integrated approach. In Bussler, C., Davies, J., Fensel, D., Studer, R., eds.: ESWS. Volume 3053 of Lecture Notes in Computer Science., Springer (2004) 76–91
12. Doan, A., Domingos, P., Halevy, A.Y.: Reconciling schemas of disparate data sources: A machine-learning approach. In: SIGMOD Conference. (2001)
13. Doan, A., Madhavan, J., Domingos, P., Halevy, A.Y.: Learning to map between ontologies on the semantic web. In: WWW. (2002) 662–673
14. Dhamankar, R., Lee, Y., Doan, A., Halevy, A.Y., Domingos, P.: imap: Discovering complex mappings between database schemas. In Weikum, G., König, A.C., Deßloch, S., eds.: SIGMOD Conference, ACM (2004) 383–394
15. Plotkin, G.D.: A note on inductive generalization. Machine Intelligence **5** (1970)
16. Maloberti, J., Sebag, M.: Fast theta-subsumption with constraint satisfaction algorithms. Machine Learning **55** (2004) 137–174