Mappings Between Domain Models in Answer Set Programming[⋆]

Stefania Costantini[1], Andrea Formisano[2], and Eugenio G. Omodeo[1]

[1] Dipartimento di Informatica, Università degli Studi di L'Aquila, L'Aquila, I-67100 Italy,
`stefcost@di.univaq.it` `omodeo@di.univaq.it`
[2] Università degli Studi di Perugia, Perugia, I-06100 Italy, `formis@dipmat.unipg.it`

Abstract. Integration of data is required when accessing multiple databases within an organization or on the WWW. Schema integration is required for database interoperability, but it is currently, a manual process.
The so-called Global-as-View approach constitutes an effective proposal for the design of data integration systems. The combination of diverse database schemata into an integrated view is achieved by means of a global conceptual data model. Consequently, the user is provided with a unified view of the data and (s)he can query such a global schema ignoring the location and structure of the data sources.
However, data integration involves resolving conflicts. Schema conflicts include naming, structural, and semantic conflicts. Then, when trying to answer a query in a data integration system, various forms of incompleteness and/or uncertainty may arise.
This work aims at demonstrating that Answer Set Programming is a suitable paradigm for defining and implementing data integration systems. In particular, we define a formalization of the Global-as-View approach which is able to cope with forms of incompleteness in data correspondence. We present a working inference engine for effectively answering queries, where each answer set corresponds to one of the possible consistent answers to the given query.

Introduction

In an emerging scenery where most applications are Web-based, and in the perspective of the Semantic Web, a novel tendency of computational logic is that of providing formalisms and tools for a context where knowledge can be shared in a distributed environment.

In the field of databases, information integration systems allow queries to be answered using a set of data sources on the WWW or across several databases in an organization. The Semantic Web should go one step further, toward an architecture where software agents can coordinate tasks using rich ontologies.

A crucial aspect where computational logic can play a role is data integration, i.e., the ability to map between different models of the same or related domains. It is unlikely

[⋆] Research partially funded by MIUR 40% project *Aggregate- and number-reasoning for computing: from decision algorithms to constraint programming with multisets, sets, and maps* and by the *Information Society Technologies programme of the European Commission, Future and Emerging Technologies* under the IST-2001-37004 WASP project.

that a global ontology or schema can be developed across several though related contexts: in practice, multiple ontologies and schemas will be developed by independent organizations, and coordination will require mappings between the different models. Such a mapping will be a set of formulae that provide the semantic relationships between the concepts in the models. If knowledge and data are to be shared, the problem of mapping between models is as fundamental as modeling itself. In current systems, mappings between models are provided manually in a labor-intensive and error-prone process.

Currently, various approaches are being developed and proposed for integrating different data sources. Also, work is under way toward a formal definition of what is a mapping between schemata [6], and which are the properties that mappings should wishfully enjoy. Informally, some basic principles are the following.

Clear semantics: The meaning of mappings should be formally defined. It should be possible to decide whether a mapping entails a certain formula. Then, mappings should be defined by a formal language, with a suitable proof system.

Managing conflicts: Data integration involves resolving conflicts. Schema conflicts include naming, structural, and semantic conflicts. Incompleteness can arise in various ways: (1) because of loss of information when the two models cover different domains, and (2) when a certain concept can be mapped between the models, but the mapping is unknown or hard to specify, and (3) when the two models cover similar domains, but the integrity constraints and/or the representations of data are different, and hence there is a gap to be filled. In real applications, cases will often arise when a mapping is incomplete, but still may suffice for the task at hand. Thus, being able to exploit incomplete mappings is one of the important issues of interest.

Allow heterogeneity: By nature, mappings between models will involve multiple representation languages. Therefore, a mapping language needs to be able to represent mappings between models in different languages.

As a first step, we consider mappings between a pair of relational-database models (i.e., schemas). In the relational context, a simple but effective approach to data integration systems through a conceptual data model is the so-called *Global-as-View* approach. In particular, the user is provided with a unified view of the data, called *global* (or 'mediated') schema. The user queries the global schema, ignoring the location and structure of the data sources, and leaving to the system the task of merging and reconciling the data at the sources. The problem of query answering in data integration systems is investigated in [2] where the global schema is expressed in terms of an extended Entity-Relationship model, and the mapping between the global schema and the sources is specified by adopting the Global-as-View approach. The problem of query answering is that when the global schema is expressed in terms of a conceptual data model, even of a very simple one, query processing becomes difficult even with the Global-as-View approach. [2] demonstrates that the problem of incomplete information arises in this case too, because it may be the case that there is more than one legal database obtainable according to the given mapping.

This work aims at demonstrating that Answer Set Programming is a suitable paradigm for defining and implementing data integration systems. In particular, we define a formalization of the Global-as-View approach which is able to cope with forms of incompleteness in data correspondence. We present a working inference engine for effectively answering queries, where each answer set corresponds to one of the possible consistent answers to the given query.

In the rest of the paper we take as leading example the one presented in [2]. However, the proposed formalization is not tailored on this example. Rather, it is composed of three parts:

- A completely general inference engine which models the correspondence between two relational schemata. The inference engine is based on an internal representation of the concepts of entities and relationships. This representation is inherently higher-order, since actual occurrences of these concepts are reified as constants. Thus, the rules composing the inference engine are actually meta-rules.
- An enhancement of the inference engine for coping with forms of incompleteness. Namely, we consider the case where it is not possible to infer/determine (through the given mapping) a unique value for some attribute of an entity or relationship. The answer to a query may thus consist of different answer sets, corresponding to consistent combinations of possible values. The inference engine can be further enhanced in a modular way to cope with more forms of uncertainty.
- A front-end interface part that maps the given global schema into the internal form.
- A back-end interface part that describes the (given) correspondence between the global schema, expressed in internal form, and the source schema(ta).

The front-end and back-end sets of formulas might be in principle automatically generated from the E-R diagrams of the global and source schema.

The proposed formalization, as it is given in Answer Set Programming, directly corresponds to an implementation. This implementation, as we will better argue below, matches the definition of *query answerability* given in [6], which is:

Definition 1. *Let $\mathcal{M}$ be a mapping between models T_1 and T_2 and an optional helper model T_3, and let Q be a query over T_1. We say that the mapping $\mathcal{M}$ enables* QUERY ANSWERABILITY *of Q if the following holds.*

Let T_2' be an extension of T_2. Let $\mathcal{I}$ be the set of interpretations of T_1 for which there exists an interpretation I_3 of T_3 and a logical model I_2 of T_2' such that $(I_1, I_2, I_3) \models \mathcal{M}$. Then, for every tuple $\bar{a}$ of constants, either $I_1 \models Q(\bar{a})$ for every $I_1 \in \mathcal{I}$ or $I_1 \not\models Q(\bar{a})$ for every $I_1 \in \mathcal{I}$.

1 The Global-as-View Approach

According to [2] a *data integration* system $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$, where $\mathcal{G}$ is the *global schema*, $\mathcal{S}$ is the *source schema*, and $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$. The source schema $\mathcal{S}$ is constituted by the schemes of the source relations. In this context we assume that the sources are expressed as relational databases. The mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ between $\mathcal{G}$ and $\mathcal{S}$ is given by associating with each concept $\mathcal{C}$ (either entity, relationship, or attribute) in the global schema a query $\mathcal{V}_C$, of the same arity as the predicate associated with $\mathcal{C}$, over the sources. No constraints are imposed on the language used to express queries in the mapping: it suffices to ensure that such a language is able to express computations over relational databases.

In order to assign semantics to a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$, [2] starts by considering a source database for $\mathcal{I}$, i.e., a database $\mathcal{D}$ for the source schema $\mathcal{S}$. Based on $\mathcal{D}$, they specify which is the *information content* of the global schema $\mathcal{G}$. We call global database for $\mathcal{I}$ any database for $\mathcal{G}$. A global database $\mathcal{B}$ for $\mathcal{I}$ is said to be *legal* with respect to $\mathcal{D}$, or, simply, legal for $\mathcal{I}$ with respect to $\mathcal{D}$, if:

- $\mathcal{B}$ is legal with respect to $\mathcal{G}$,
- for each element e of $\mathcal{G}$, the set of tuples $e^{\mathcal{B}}$ that $\mathcal{B}$ assigns to e is coherent with the set of tuples computed by the associated query $\mathcal{V}_e$ over $\mathcal{D}$, i.e., $\mathcal{V}_e^{\mathcal{D}} \subseteq e^{\mathcal{B}}$.

The above definition implies that sources are regarded as being sound: the data they provide to the integration system satisfy the global schema, but are not necessarily complete.

The semantics of a data integration system $\mathcal{I}$ with respect to a source database $\mathcal{D}$ for $\mathcal{I}$ is the set of global databases that are legal for $\mathcal{I}$ with respect to $\mathcal{D}$. Given a source database $\mathcal{D}$, different situations are possible. The first case is when no legal global database exists. This happens, in particular, when the data at the sources retrieved by the queries associated with the elements of the global schema do not satisfy the functional attribute constraints. The second case occurs when several legal global databases exist. This happens, for example, when the data at the sources retrieved by the queries associated to the global relations do not satisfy the IS_A relationships of the global schema. In this case, it may happen that several ways exist to add suitable objects to the elements of $\mathcal{G}$ in order to satisfy the constraints. Each such way yields a legal global database. The problem of incomplete information in the Global-as-View approach is overlooked in traditional data integration systems, which either express the global schema as a set of plain relations, or consider the sources as being exact.

An algorithm for computing the set of certain answers to queries posed to a data integration system can be found in [2]. The key feature of this algorithm is to reason about both the query and the conceptual global schema in order to infer which are the certain answers to the query. Indeed, the authors observe that a simple unfolding strategy does not work, and several complications arise due to the potential existence of different legal databases. In the following sections, we will show that Answer Set Programming allows one to define a mapping representation language and an inference engine that works fine also in presence of different legal databases, thanks to the very nature of the formalism.

2 Answer Set Programming

Answer Set Programming (ASP) is an emerging paradigm of logic programming [7,5] based on the Answer Set (or equivalently Stable Model) semantics [3,4]: each solution to a problem is represented by an answer set (also called a Stable Model) of a deductive database/function–free logic program encoding the problem itself. The Answer Set semantics is a view of logic programs as sets of inference rules (more precisely, default inference rules). Alternatively, one can see a program as a set of constraints on the solution of a problem, where each answer set represents a solution compatible with the constraints expressed by the program. Consider for instance the simple program $\{q \leftarrow \textit{not}\ p.\ \ p \leftarrow \textit{not}\ q.\}$: the first rule is read as "assuming that p is false, we can *conclude* that q is true." This program has two answer sets. In the first one, q is true whereas p is false; in the second one, p is true whereas q is false. Then, in ASP we are able to manage cyclic negative dependencies that represent incomplete knowledge or denote the possibility of different alternatives, by representing each consistent choice by means of an answer set. Consequently, a program in ASP may have none, one, or several answer sets.

To solve a problem using ASP means to write a logic program whose answer sets correspond to solutions, and then find a solution using an answer set solver [1]. The basic approach to writing such a program is known as the "generate-and-test" strategy. First one writes a group of rules for defining "potential solutions" i.e., an easy-to-describe superset of the set of solutions. Then one adds a group of constraints that rule out the potential solutions that are not solutions.

In this paper we adopt the syntax of the SMODELS solver, that we briefly describe below, referring to an answer set program for finding the three-colorings of a graph. The statement node(0..3) is a shortcut for the definition of a set of facts, namely node(0),. . . ,node(3). The symbol "|" denotes exclusive disjunction (which is syntactic sugar, since it might be expressed through negation), and allows us to state that a node can be assigned one of the three colors red, blue, green, that are introduced in the facts col(·). This defines all possible colorings of the graph. It remains to be stated that we wish to select only the colorings where adjacent nodes have a different color.

The rule with empty head is a *constraint*, whose conditions cannot be all true, otherwise they would imply falsity. It is again syntactic sugar, since a constraint, say :- a, not b, c, can be rephrased as f :- not f, a, not b, c, where f is a fresh atom not appearing elsewhere in the program: in fact, in the Answer Set semantics no true atom can be supported by the negation of another true atom, and in particular atom f cannot be supported by its own negation; consequently either f is forced to be false, or there is no answer set; for f to be false, some of the other conditions must be false, i.e., the conditions cannot be all true. In particular, in this program the constraint states that there cannot be two adjacent nodes (nodes connected by an edge) to which the same color is assigned.

In alternative, one can use the following rule with head in brackets, called *weight constraint*, which means that for all X which is a node, the property color(X,C) can take one and only one value for C, among those defined by the predicate col(C).

```
1{color(X,C): col(C)}1 :- node(X).
```

The general form of such a kind of clauses is

$$n\{\langle property\_def\rangle{:}\langle range\_def\rangle\}m \text{ :-}\langle search\_space\rangle$$

where: the conditions $\langle search\_space\rangle$ in the body define the set of objects of the domain to be checked; the atom $\langle property\_def\rangle$ in the head defines the property to be checked; the conjunction $\langle range\_def\rangle$ defines the possible values that the property may take on the objects defined in the body, namely by providing a conjunction of k unary predicates each defining a range for one of the k variables that occur in $\langle property\_def \rangle$ but not in $\langle search\_space \rangle$; n and m are the minimum and maximum number of values that the specified property may take on the specified objects.

Finally, hide P is a directive whose obvious meaning is that of instructing the solver to omit atoms concerning P when returning the answer sets. Namely, the answer sets of this program give all three-colorings of the given graph.

```
col(red).      col(blue).    col(green).
node(0..3).   edge(0,1).   edge(1,2).   edge(2,0).   edge(2,3).   edge(1,3).
color(X,red) | color(X,blue) | color(X,green) :- node(X).
:- edge(X,Y), col(C), color(X,C), color(Y,C).
1{color(X,C): col(C)}1 :- node(X).
hide node(X).   hide edge(X,Y).   hide col(C).
```

When fed with this program, SMODELS (and, with similar syntax, any of the solvers) will give the answers:

```
Answer: 1.  Stable Model:  color(0,green)  color(1,blue)  color(2,red)  color(3,green)
Answer: 2.  Stable Model:  color(0,blue)  color(1,green)  color(2,red)  color(3,blue)
...
Answer: 6.  Stable Model:  color(0,blue)  color(1,red)  color(2,green)  color(3,blue)
```

where each answer set corresponds to one of the possible 3-colorings of the graph.

The reason why this approach is particularly well suited for representing mappings between data models is exactly that the query answering problem can be coped with also in the presence of incomplete/ambiguous knowledge by making all the different possible answers to queries explicit.

3 The Global-as-View Approach in Answer Set Programming

In this section, we implement the Global-as-View approach and a query-answering algorithm in Answer Set Programming along the lines of [6]. In particular, we define a helper model with suitable meta-level aspects, so as to define an inference engine that models all the basic features of relational databases, thus enforcing query answerability.

In what follows, we assume that the global schema is expressed by means of an (extension of the) Entity-Relationship modeling language. We assume also the local/source schema to be a relational schema.

Front-end. The formalization of the relational model in Answer Set Programming is based on an internal description of all the concepts involved, in the following form:

entity(E,X), relat(R,X1,. . . ,Xk), attr(A,X,Y), and schema(R,E1,. . . ,Ek):

- an entity is described by the predicate entity(E,X), where E is an integer number and X is intended to range over the instances of the specific entity. Roughly speaking, E should be intended as an "internal name" of a given entity in the formalization.

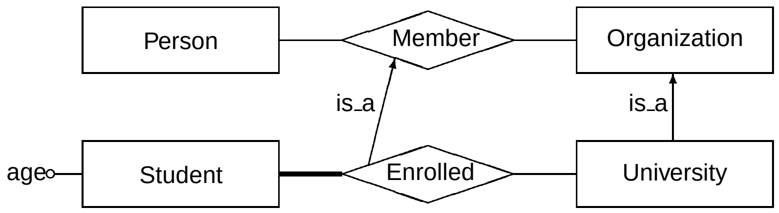

Fig. 1. Example of global schema

Person$(x) \leftarrow s_1(x)$	Student$(x) \leftarrow s_3(x,y) \vee s_4(x,z)$
Organization$(x) \leftarrow s_2(x)$	University$(x) \leftarrow s_5(x)$
Member$(x,y) \leftarrow s_7(x,z), s_8(z,y)$	Enrolled$(x,y) \leftarrow s_4(x,y)$
age$(x,y) \leftarrow s_3(x,y) \vee s_6(x,y,z)$	

Fig. 2. Sample mapping for the global schema of Figure 1

- Any k-ary relationship is described by the predicate relat(R,X1,...,Xk). As before, R is an integer number identifying the specific relationship, while X1,...,Xk range over the instances of the entities participating in the relationship.
- Any attribute is named by an integer number A. Its value (Y) corresponding to a specific instance X of the entity is assigned by means of the fact attr(A,X,Y).
- The predicate schema(R,E1,...,Ek) asserts that entities E1,...,Ek participate in the relationship R. (In this case R, E1,...,Ek are integer numbers identifying relationships and entities.)

Some auxiliary predicates can be used to designate the integer numbers to be used as names for objects, for instance:

```
e(1..5).    % range for entity numbers
r(1..5).    % range for relation numbers
a(1..5).    % range for attribute numbers
```

The use of predicates for defining concepts of the relational model, and of reification (constants, namely integer numbers) for specific instances of concepts makes the internal formalization inherently higher-order. The inference engine we will present below composes of meta-rules coping with general concepts, that will be then instantiated on the case at hand. The approach presents valuable advantages concerning elaboration-tolerance:

- the rules are defined for classes of constructs, thus they are generally applicable;
- this makes it easier to cope with changes in the format of global schema.

For demonstrating the approach on a practical case, we take the example proposed in [2], represented by the schema in Figure 1. This will be our working example in what follows. Through it, we will illustrate our ASP-based approach.

Example 1. Figure 1 shows the global schema $\mathcal{G}_1$ of a data integration system $\mathcal{I}_1 = \langle \mathcal{G}_1, \mathcal{S}_1, \mathcal{M}_1 \rangle$, where age is a functional attribute, Student has a mandatory participation in the relationship Enrolled,[1] Enrolled is-a Member, and University is-a Organization. The schema models persons who can be members of one or more organizations, and students who are enrolled in universities. Suppose that $\mathcal{S}_1$ is constituted by s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, and that the mapping $\mathcal{M}_1$ can be defined as in Figure 2.

The redefinition of the global schema of Figure 1 in terms of our proposed (meta-)formalization consists in the following rules:

```
person(X):- entity(1,X).
organization(X):- entity(2,X).
member(X,Y):- schema(1,E1,E2),entity(E1,X),entity(E2,Y),relat(1,X,Y).
student(X):- entity(3,X).
university(X):- entity(4,X).
enrolled(X,Y):- schema(2,E1,E2),entity(E1,X),entity(E2,Y),relat(2,X,Y).
age(X,Y):- val(X),val(Y),attr(1,X,Y).
```

We also the logical definition of global schema (it specifies domain and codomain of relationships in terms of entity numbers):

```
schema(1,1,2).  % entities 1 (person) and 2 (organization) participate in relationship 1 (member)
schema(2,3,4).  % entities 3 (student) and 4 (university) participate in relationship 2 (enrolled)
```

□

The representation is pretty general, and for practical cases might be automatically generated from E-R diagrams. The internal form we have defined plays the role of a ”helper model” on which the inference engine can be defined neatly and generally. The back-end part describes the correspondence (that, in the Global-as-View approach, is supposed to be given) between the global schema expressed in the internal form, and the source schema.

Back-end: Correspondence between global schema and source schema. The correspondence between the global schema (in abstract form, i.e., through the helper schema) and the source schema is given by means of suitable clauses that establish a connection between the objects of the helper model and the data sources.

Example 2. Consider our working example (cf. Figure 2). This is the rendering of the global schema through the helper schema:

```
entity(1,X):- val(X),s(1,X).
entity(2,X):- val(X),s(2,X).
relat(1,X,Y):- val(X),val(Y),val(Z),s(7,X,Z),s(8,Z,Y).
entity(3,X):- val(X),val(Y),s(3,X,Y).
   % student (i.e. entity 3) is described implicitly as the domain of an unknown relationship
entity(3,X):- val(X),val(Z),s(4,X,Z).
   % student is described implicitly as the domain of enrolled
entity(4,X):- val(X),s(5,X).
relat(2,X,Y):- val(X),val(Y),s(4,X,Y).
attr(1,X,Y):- val(X),val(Y),s(3,X,Y).
attr(1,X,Y):- val(X), val(Y), val(Z), s(6,X,Y,Z).
```

□

[1] Mandatory participations are depicted by using thicker lines.

We assume that the instance of the source schema is given by using the predicates val and isValue$n$, in terms of a number of asserted facts in the ASP program. In particular, val describes all the values occurring in the instance of the source database under consideration. The predicate isValue1 associates such values to specific entities of the source schema (similarly, isValue2 and isValue3, manage values for source relations of arity 2 and 3, respectively).

Example 3. This is a description of a possible instance for the source schema of Example 1 and Figure 2:

```
source(1..8).
s(N,A):- source(N), isValue1(N,A).
s(N,A,B):- source(N), isValue2(N,A,B).
s(N,A,B,C):- source(N), isValue3(N,A,B,C).
isValue1(1,p1).    isValue1(1,p2).      isValue1(2,o1).
isValue1(5,u1).    isValue1(5,u2).      isValue1(5,u3).    isValue1(5,u4).
isValue2(4,t1,u1). isValue2(4,t2,any4). isValue2(8,i1,o1). isValue2(8,i2,u1).
isValue2(7,p1,i1). isValue2(7,p2,i2).   isValue2(7,p3,i3).
```

These facts represent the following instance of the relational source schema: the relation s_1 contains the tuples p1 and p2; s_2 contains the tuple o1; s_5 contains the tuples u1, u2, u3, and u4; s_4 contains the tuples (t1, u1) and (t2, a1) (notice that a1 is not in s_5); and so on. In order to make the grounding process performed by SMODELS possible, we need to list (at least) all possible values occurring in the source instance:

```
val(c1). val(c2). val(c3).   val(c4).             val(s1). val(s2).  val(s3). val(s4).
val(e1). val(e2). val(e3).   val(e4).             val(i1). val(i2).  val(i3).
val(p1). val(p2). val(p3).   val(p4).             val(o1). val(o2).
val(u1). val(u2). val(u3).   val(u4).             val(t1). val(t2).  val(t3). val(t4).
val(a1).          val(any4). anyval(4, any4).              val(null).
```

□

The Inference Engine. The inference engine declaratively describes how queries should be answered and constraints should be imposed, in terms of the proposed formalization. Procedurally, it allows queries to be answered. Differently from other proposals, the approach based on ASP permits a correct handling of information incompleteness and provides the user with all possible legal global databases.

The use of a naming of the objects of the relational schema at hand allows us to tersely describe integrity constraints. For instance, the following two clauses define left and right projections for each relationship in the model (notice that we restrict our treatment to dyadic relationships in the global schema, the generalization to generic arity is easy):

```
proj_relat1(R,X):- schema(R,E1,E2),entity(E1,X),entity(E2,Y),relat(R,X,Y).
proj_relat2(R,Y):- schema(R,E1,E2),entity(E1,X),entity(E2,Y),relat(R,X,Y).
```

Analogously, IS_A relations, both for entities and for relationships, are easily asserted by explicitly referring to the names in helper model: assertion is_a_r(R1,R2) means relationship R1 IS_A R2. Similarly, an assertion of the form is_a_e(E1,E2) means entity E1 IS_A E2:

We also introduce suitable meta-rules for IS_A chaining in the helper model:

```
relat(M,X,Y):- schema(N,E1,E2), entity(E1,X), entity(E2,Y), is_a_r(N,M), relat(N,X,Y).
entity(M,X):- val(X), is_a_e(N,M), entity(N,X).
```

The treatment of mandatory participation and functionality constraints is also immediate. For example, mandatory participations is asserted by means of two predicates: assertion mandatory1(E1,R) (respectively, mandatory2(E2,R)) means that entity E1 (resp., E2) must participate in a relationship R as the first (resp., second) component. These assertions are handled by suitable clauses, namely:

```
:- entity(E,V), schema(R,E,E1), entity(E1,V1), mandatory1(E,R), not proj_relat1(R,V).
:- entity(E,V), schema(R,E1,E), entity(E1,V1), mandatory2(E,R), not proj_relat2(R,V).
```

whose aim is to ensure that whenever an entity E has a mandatory participation in a relationship R as first/second argument, each value of E must belong to the left/right projection of R. Consequently, no illegal database, i.e., answer set violating the constraint, can be produced.

Example 4. In our working example we assert:

```
is_a_r(2,1). % enrolled IS_A member
is_a_e(3,1). % student IS_A person
is_a_e(4,2). % university IS_A organization
mandatory1(3,2). % mandatory participation of student in enrolled
```

□

Integrity constraints for functionality are imposed by assertions of the form functional1(R) (resp. functional2(R)). This assertion, as expected, says that the relationship R must be single-valued on its first (resp. second) argument (clearly, the treatment can be generalized to describe multi-functions with bounds on the cardinality of images by exploiting weight-constraints). The following clauses express functionality constraints. Hence, no illegal database, i.e., answer set violating the constraint, can be produced.

```
:- schema(R,E1,E2), functional1(R), entity(E1,V1), entity(E1,V2),
   V1 != V2, proj_relat1(R,V1), proj_relat1(R,V2).
:- schema(R,E1,E2), functional2(R), entity(E2,V1), entity(E2,V2),
   V1 != V2, proj_relat2(R,V1), proj_relat2(R,V2).
```

Incompleteness of information. Consider a mapping $\mathcal{M}$ between a global schema $\mathcal{G}$ and a source schema $\mathcal{S}$. It may be the case that more than one legal global database exists w.r.t. a specific global query. In particular, this may happen when an IS_A relation imposed at the global level does not reflect an analogous property of the source schema (notice that this is the case for our working example). Consequently, there may be several manners to (correctly) answer a global query. It may be necessary to add suitable objects to the global database in order to satisfy the IS_A constraint. Such an "extension" of the answer is somehow arbitrary: because only incomplete information can be drawn from the source schema, there may be no unique choice for such "extra" objects. In other words, the query-answering process should be able to fill the gap between global and source models by providing all legal global databases which are coherent with the (partial) information available at the source level. As we will see, this is one of the achievements of our approach.

Example 5. Consider the mapping of Figure 2 and a source database where s_4 stores (t1,u1) and (t2,a1), while u1, u2, u3, and u4 are the only tuples stored in s_5. The global schema (cf. Figure 1) imposes a mandatory participation of Student to the relationship Enrolled. By extracting information from the source schema through the mapping, we can only discover that:

- t1 is enrolled in u1;
- t2 should be enrolled. However, it is unknown which is the corresponding university;
- There are four distinct known universities: u1, u2, u3, and u4.

By assuming that any specific student can be enrolled in at most one university (i.e., Enrolled is functional), we would like this knowledge to be so extended as to associate t2 to one of the known universities. □

The use of ASP as inference framework for modeling mappings and answering global queries allows the user to obtain all possible legal global databases which are coherent with the information extracted from the source level. This aim is achieved by introducing (in the helper model) a specific object (denoted by any) that represents an unknown value.

The following clauses manage the special value any and characterize all extensions of a relationship R obtainable by considering any of the members of the domain/codomain entity. In particular, the use of weight-constraints ensures that all these extensions are obtained by adding a single object to the (global) relationship. Notice that, in order to avoid confusion among couples of entities involved in different relationships, each entity must have an associated any value, to be used only in the context of that relationship. To this aim, we are able to associate the needed any values to entities by means of assertions of the form anyval(E2,A).

```
1{vals2(R,Y):val(Y)}1:- val(X),schema(R,E1,E2),relat(R,X,A),anyval(E2,A),entity(E2,Y).
1{vals1(R,Y):val(X)}1:- val(Y),schema(R,E1,E2),relat(R,A,X),anyval(E1,A),entity(E1,X).
relat(R,X,Y):- schema(R,E1,E2),entity(E1,X),vals1(R,X),relat(R,A,Y),anyval(E1,A),entity(E2,Y).
relat(R,X,Y):- schema(R,E1,E2),entity(E1,X),vals2(R,Y),relat(R,X,A),anyval(E2,A),entity(E2,Y).
```

The Approach at Work. Let us conclude our working example by illustrating how the inference engine of SMODELS handles queries in presence of incomplete information.

Example 6. Below we report the answer sets produced by SMODELS for the above source instance. Notice that in such data source there exists an object occurring in the relationship s_4 (namely, a1) which corresponds to t2. Since in the global schema the participation of Student to the relationship Enrolled is mandatory, the mapping should identify an object in University related to t2 through Enrolled. This knowledge cannot be obtained from the source schema: the only known fact is that there exist four universities in the universe of discourse. (This is asserted by the facts isValue1(5,u1), isValue1(5,u2), isValue1(5,u3), and isValue1(5,u4).) Consequently, any answer set where t2 is enrolled in one of these universities must be considered as legal. SMODELS produces four different answer sets. We list below one of them, together with the differences between it and the others (underlined facts):

```
 Answer: 1.  Stable Model:   mandatory1(3,2)   enrolled(t1,u1)   enrolled(t2,u3)
   university(u4)  university(u1)  university(u2)  university(u3)  student(t2)  student(t1)
   member(p1,o1)   member(p2,u1)   member(t1,u1)   member(t2,u3)
   organization(u4)  organization(o1)  organization(u1)  organization(u2)  organization(u3)
   person(t2)   person(p1)   person(p2)   person(t1)

Answer: 2.  Stable Model:  . . .  enrolled(t2,u4)  . . .  member(t2,u4)  . . .
Answer: 3.  Stable Model:  . . .  enrolled(t2,u2)  . . .  member(t2,u2)  . . .
Answer: 4.  Stable Model:  . . .  enrolled(t2,u1)  . . .  member(t2,u1)  . . .                □
```

4 Remarks on the Approach

It is not hard to believe that the rules that define the global schema in terms of the helper model can be generated automatically, given the E-R schema or any other semi-formal sufficiently expressive representation. The rules that state the correspondence between the global schema and the source schema through the helper model can also be generated automatically.

In real applications however, one cannot suppose to incorporate the source database into the program like we have done for the simple example we have illustrated (cf., for instance, the facts of the form val(·)). Rather, access to the source database should in perspective be performed through a suitable interface.

The core of our approach is the internal formalization, or in other terms the helper model, which, once equipped with suitable interfaces, allows heterogeneity, and, as we have seen, is able to cope with incompleteness.

In our approach, it is possible to give a definition of mapping satisfaction [6] as follows, where let T_{ASP} be the helper model we have defined before, and $P_{\mathcal{M}}$ be the ASP program, suitably customized to the mapping at hand.

Definition 2. *Let $\mathcal{M}$ be a mapping between the models T_1 and T_2, and assume that the helper model $T_3 = T_{ASP}$ is a correct formalization of $\mathcal{M}$. Let I be an interpretation of the corresponding ASP program $P_{\mathcal{M}}$. Then, I satisfies the mapping $\mathcal{M}$ if and only if I is an answer set of $P_{\mathcal{M}}$.*

For proving query answerability we have to prove the correctness of our inference engine, and this is for the time being just a claim. The reader however is invited to check this claim by running the above ASP program, on the proposed example as well as on others.

References

[1] Web locations of the best known ASP solvers.
CCALC: *http://www.cs.utexas.edu/users/mcain/cc*
Cmodels: *http://www.cs.utexas.edu/users/tag/cmodels.html*
DeReS: *http://www.cs.engr.uky.edu/~lpnmr/DeReS.html*
DLV: *http://www.dbai.tuwien.ac.at/proj/dlv/*

NoMoRe: *http://www.cs.uni-potsdam.de/ linke/nomore/*
SMODELS: *http://www.tcs.hut.fi/Software/smodels/*

[2] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. *Accessing data integration systems through conceptual schemas*. In: Proc. of the 10th Italian Conf. on Database Systems (SEBD'02), 2002.

[3] M. Gelfond and V. Lifschitz. *The stable model semantics for logic programming*. Proc. of 5th ILPS conference : 1070–1080, 1988.

[4] M. Gelfond and V. Lifschitz. *Classical negation in logic programs and disjunctive databases*. New Generation Computing: 365–387, 1991.

[5] V. Lifschitz. *Answer Set Planning.* In: D. De Schreye (ed.) Proc. of the 1999 International Conference on Logic Programming ICLP'99 (invited talk), The MIT Press, pp. 23–37, 1999.

[6] J. Madhavan, P. A. Bernstein, P. Domingos, and A.Y. Halevy. *Representing and Reasoning About Mappings between Domain Models*. In: Proc. 18th National Conference on Artificial Intelligence (AAAI 2002), Edmonton, Canada.

[7] W. Marek and M. Truszczyński. *Stable models and an alternative logic programming paradigm*, The Logic Programming Paradigm: a 25-Year Perspective, Springer-Verlag, pp. 375–398, 1999.