D2I

Integrazione, Warehousing e Mining di sorgenti eterogenee

Programma di ricerca (cofinanziato dal MURST, esercizio 2000)

Methodology and Tools to Reconcile Data

Diego Calvanese, Domenico Lembo, Maurizio Lenzerini

D1.R11 **30 novembre 2001**

Sommario

A data integration system (DIS) provides access to a set of heterogeneous data sources through a so-called global schema. There are basically two approaches for designing a DIS. In the *global-as-view* (GAV) approach, one defines the elements in the global schema as views over the sources, whereas in the *local-as-view* (LAV) approach, one characterizes the sources as views over the global schema. In this paper we propose methodologies to reconcile data, both for LAV and GAV. For LAV, we propose to declaratively specify reconciliation correspondences to be used to solve conflicts among data in different sources, and define an algorithm that rewrites queries posed on the global schema in terms of both the source elements and the reconciliation correspondences. For GAV, it is a common opinion that query processing is much easier than in LAV, where query processing is similar to query answering with incomplete information. However, we show that, when constraints are expressed over the global schema, the problem of incomplete information arises in GAV as well. We provide a general semantics for a GAV DIS, and specify algorithms for query answering in the presence of both incompleteness of the sources and inconsistencies between the data at the sources and the constraints on the global schema.

Tema	Tema 1: Integrazione di dati provenienti da sorgenti eterogenee
Codice	D1.R11
Data	30 novembre 2001
Tipo di prodotto	Rapporto Tecnico
Numero di pagine	35
Unità responsabile	RM
Unità coinvolte	RM
Autore da contattare	Domenico Lembo Dipartimento di Informatica e Sistemistica Università di Roma “La Sapienza” Via Salaria 113, 00198 Roma, Italia `lembo@dis.uniroma1.it`

Methodology and Tools to Reconcile Data

Diego Calvanese, Domenico Lembo, Maurizio Lenzerini

30 novembre 2001

Abstract

A data integration system (DIS) provides access to a set of heterogeneous data sources through a so-called global schema. There are basically two approaches for designing a DIS. In the *global-as-view* (GAV) approach, one defines the elements in the global schema as views over the sources, whereas in the *local-as-view* (LAV) approach, one characterizes the sources as views over the global schema. In this paper we propose methodologies to reconcile data, both for LAV and GAV. For LAV, we propose to declaratively specify reconciliation correspondences to be used to solve conflicts among data in different sources, and define an algorithm that rewrites queries posed on the global schema in terms of both the source elements and the reconciliation correspondences. For GAV, it is a common opinion that query processing is much easier than in LAV, where query processing is similar to query answering with incomplete information. However, we show that, when constraints are expressed over the global schema, the problem of incomplete information arises in GAV as well. We provide a general semantics for a GAV DIS, and specify algorithms for query answering in the presence of both incompleteness of the sources and inconsistencies between the data at the sources and the constraints on the global schema.

1 Introduction

Data integration is the problem of combining the data residing at different sources, and providing the user with a unified view of these data, called global (or, mediated) schema [42]. The interest in data integration systems has been continuously growing in the last years. Many organizations face the problem of integrating data residing in several sources. Companies that build a Data Warehouse, a Data Mining, or an Enterprise Resource Planning system must address this problem. Also, integrating data in the World Wide Web is the subject of several investigations and projects nowadays. Finally, applications requiring accessing or re-engineering legacy systems must deal with the problem of integrating data stored in pre-existing sources.

The typical architecture of an integration system is described in terms of two types of modules: wrappers and mediators [56, 53]. The goal of a wrapper is to access a source, extract the relevant data, and present such data in a specified format. The role of a mediator is to collect, clean, and combine data produced by different wrappers (or mediators), so as to meet a specific information need of the integration system. In the design of an integration system, one of the core problems is the specification and the realization of mediators. Such activity is a very complex task, which comprises several different issues [38].

An important aspect is the specification of the mapping between the global schema, which provides the interface by which users issue their queries to the system, and the sources, which contain the data needed to answer such queries. By exploiting such a mapping the system can access the appropriate sources and answers the queries, thus freeing the user from the knowledge on where data are, and how data are structured at the sources. Two basic approaches have been used to specify the mapping between the sources and the global schema [42, 43, 45]. The first approach, called *global-as-view* (or simply GAV), requires that the global schema is expressed in terms of the data sources. More precisely, to every element of the global schema, a view over the data sources is associated, so that its meaning is specified in terms of the data residing at the sources. The second approach, called *local-as-view* (LAV), requires the global schema to be specified independently from the sources. In turn, the sources are defined as views over the global schema. The relationships between the global schema and the sources are thus established by specifying the information content of every source in terms of a view over the global schema. Intuitively, the GAV approach provides a method for specifying the data integration system with a more procedural flavor with respect to the LAV approach. Indeed, whereas in LAV the designer of the data integration system may concentrate on specifying the content of the source in terms of the global schema, in the GAV approach, one is forced to specify how to get the data of the global schema by queries over the sources.

A further issue is the choice of the method for computing the answer to queries posed in terms of the global schema, which is one of the most important problems in the design of a data integration system. For this purpose, the system should be able to reformulate the query in terms of a suitable set of queries posed to the sources. In this reformulation process, the crucial step is deciding how to decompose the query on the global schema into a set of subqueries on the sources, based on the meaning of the mapping. The computed subqueries are then shipped to the sources, and the results are assembled into the final answer. It is well known that processing queries in the local-as-view approach is a difficult task [49, 53, 32, 1, 30, 16, 18]. Indeed, in this approach the only knowledge we have about the data in the global schema is through the views representing the sources, and such views provide only partial information about the data. Therefore, extracting information from the data integration system is similar to query answering with incomplete information, which is a complex task [54]. On the other hand, query processing is considered much easier in the global-as-view approach, where in general it is assumed that answering a query basically means unfolding its atoms according to their definitions in terms of the sources [34]. The reason why unfolding does the job is that the global-as-view mapping essentially specifies a single database satisfying the global schema, and evaluating the query over this unique database is equivalent to evaluating its unfolding over the sources.

Independently of the mapping between the sources and the global schema, and the method used to answer queries, one needs to take into account that the necessary data are provided by different, autonomous, and heterogeneous sources, and hence need to be interpreted, combined, and reconciled. Interpreting data can be regarded as the task of casting them into a common representation. Moreover, the data returned by the various sources need to be combined to provide the information requested to the integration system. The complexity of this step is due to several problems [28], such as

- possible representation mismatches between data referring to the same real world object;
- possible errors in the data stored in the sources;
- possible incompleteness in the data stored in the sources;
- possible inconsistencies between values representing the properties of the real world objects in different sources.

As a consequence, it becomes necessary to provide means to reconcile and merge the data coming from different sources, so as to resolve the inconsistencies among values, and also to make use of all the information available globally to possibly overcome incompleteness and lack of data from the sources.

In commercial environments for design and management of integration systems, the above tasks are taken care of through ad hoc components [40, 34]. In general, such components provide the user with the capability of specifying the mapping between the sources and the global schema by browsing through a meta-level description of the relations of the sources. In addition, they generally provide both for automatic code generators and for the possibility of attaching procedures to accomplish ad hoc transformations and filtering of the data. Even though there are powerful and effective environments with the above features, their nature is inherently procedural, and they typically require ad-hoc intervention on a query-by-query basis.

Several recent research contributions address these problems from a more formal perspective [39, 35, 55, 33, 58, 59, 48, 29]. Generally speaking, these works follow the global-as-view approach. Research projects concerning the local-as-view approach have concentrated on the problem of reformulating a query in terms of the source relations [44, 50, 52, 3, 25, 6, 32, 26, 22, 31, 1, 30, 15, 16, 17]. However, none of them addresses both the problem of data reconciliation at the instance level, and the problem of reformulating queries posed to an integration system via a global schema, properly taking into account not only the mapping between the global schema and the sources, but also the constraints imposed by the global schema itself, and the inconsistencies and mismatches between data in different sources.

In this paper we present different methodologies for data integration and reconciliation, suitable for the local-as-view and the global-as-view approach, respectively. In both cases we adopt a framework in which the information content of the integration system, represented in the global schema, is expressed in an Entity-Relationship formalism. As a consequence, such a global schema may express several forms of constraints at the global level.

- For the local-as-view approach, we propose to declaratively specify various types of reconciliation correspondences between data in different sources. Three types of *Reconciliation Correspondences* are taken into account, namely Conversion, Matching, and Merging Correspondences. Such correspondences are used to support the task of specifying the correct mediators for the integration system. For this purpose, we propose a methodology that relies on a query rewriting algorithm, whose role is to reformulate queries posed to the integration system in terms of both the source relations and the Reconciliation Correspondences. The characteristic feature of the algorithm is that it takes into account the constraints imposed by the global schema, and uses the Reconciliation Correspondences for cleaning, integrating, and reconciling data coming from different sources.

- For the global-as-view approach, we show that during query processing, one needs to take into account the semantics of the global schema to reconcile the answers coming from different sources with the global constraints holding for the integration system. Indeed, the presence of constraints on the global schema makes query processing more involved than in the simplified framework usually considered in the literature. In particular, we show that the semantics of a data integration system is best described in terms of a set of databases, rather than a single one, and this implies that, even in the global-as-view approach, query processing is intimately connected to the notion of querying *incomplete databases*. We then formalize the notion of correct answer in a data integration system in which the global schema contains constraints. We also present, for the case at hand (i.e., where the global schema is expressed in the Entity-Relationship model) a query processing strategy that is able to provide all correct answers to a query posed to the system.

  When we consider also functional attributes in the global schema, the problem arises of reconciling data coming from different sources that violates functionality. For this case, we propose to formalize the answer to a query by weakening the first-order semantics, and show how the query answering algorithm can be encoded in disjunctive DATALOG. We show also that in this case the data complexity of computing the answer is no longer polynomial but coNP-complete.

The paper is organized as follows. In Section 2, we present the conceptual model, which is at the basis of the integration framework adopted in the paper. Such a framework is introduced in Section 3. The problem of data reconciliation is addressed in Section 4 in the local-as-views setting, and in Section 5 in the global-as-view setting. Section 6 concludes the paper.

2 The Conceptual Data Model

The conceptual model, which is at the basis of the integration framework introduced in the next section, is the basic *Entity-Relationship* (ER) model [21], extended with various forms of constraints [4].

Formally, an *ER schema* is a collection of entity, relationship, and attribute definitions over an *alphabet $\mathcal{A}$ of symbols*. Entities, relationships and attributes are called the *concepts* of the ER schema. An *entity set* (or simply *entity*) denotes a set of objects, called its *instances*, that have common properties. Properties that are due to relations to other entities are modeled through the participation of the entity in relationships. A *relationship set* (or simply *relationship*) denotes a set of tuples (also called its instances), each of which represents an association among a different combination of instances of the entities that participate in the relationship. Since each entity can participate in a relationship more than once, the notion of *ER-role* is introduced, which represents such a participation and to which a unique name is assigned. The *arity* of a relationship is the number of its ER-roles. We do not use names for referring to ER-roles, rather, we simply use the numbers corresponding to their positions in the relationships. *Cardinality constraints* can be attached to an ER-role in order to specify the number of times each instance of an entity is allowed to participate via that ER-role in instances of the relationship. In particular, such constraints can be used to indicate that instances of an entity must necessarily participate in instances of a certain relationship in which the entity is involved (in this case the minimum cardinality is 1). We call these constraints *mandatory participation* constraints. Elementary properties are modeled through *attributes*, whose values belong to one of several predefined domains, such as `Integer`, `String`, or `Boolean`, or *abstract domains* that allow the designer to distinguish between the different meanings that values of concrete domains may have. Notice that in our model each attribute is associated to a unique entity or relationship, i.e., different entities and relationships have disjoint sets of attributes. Furthermore, cardinality constraints can be attached to an attribute in order to indicate the number of times each instance of an entity (relationship) has associated a value for such attribute. In particular, a cardinality constraint can indicate that an attribute A of an entity E is *mandatory*, i.e., each instance of E must necessarily be associated to a value of A (in this case the minimum cardinality is 1), or *functional*, i.e., an instance of E cannot be associated to more than one value of A (in this case the maximum cardinality is 1).

The semantics of an ER schema is formally defined by specifying when a database satisfies all constraints imposed by the schema. For the sake of simplicity we consider only databases defined over a fixed (infinite) alphabet Γ, disjunct from $\mathcal{A}$, of symbols, each one denoting a semantic value. A given database $\mathcal{B}$ assigns to each entity a subset of Γ, to each attribute of an entity a binary relation over Γ, to each relationship R of arity n, a set of n-tuples of elements of Γ, and to each attribute of a relationship of arity n an $(n+1)$-ary relation over Γ. The set of objects assigned by $\mathcal{B}$ to an entity, attribute, or relationship is called the set of its *instances* in $\mathcal{B}$. We say that $\mathcal{B}$ is *coherent* with respect to an ER schema $\mathcal{C}$ if it satisfies all the constraints in $\mathcal{C}$, i.e.:

- a cardinality constraint (n, m) associated to the participation of an entity E to a relationship R via the ER-role i is satisfied if each instance of E appear at least n times and at most m times as component i in the instances of R. In particular, if the cardinality constraint indicates a mandatory participation of E in R, it is satisfied if each instance of E appear at least one time as component i in the instances of R;

- a cardinality constraint (n, m) attached to an attribute A of an entity E (or relationship R) is satisfied if each instance of E (resp. R) is associated to at least n, and at most m values of A. In particular, if the cardinality constraint indicates that A is mandatory, it is satisfied if each instance of E (resp. R) is associated to at least one value of A. In the

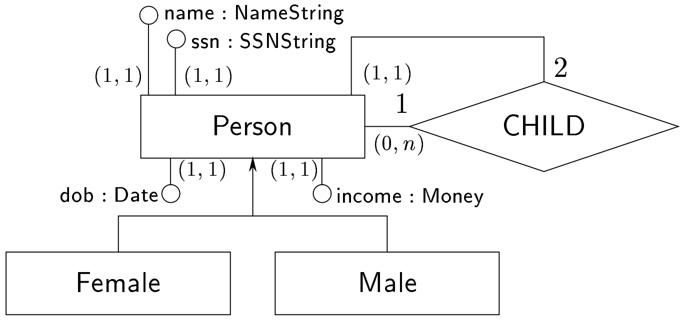

Figure 1: Entity-Relationship schema for parent-child relationship

case that the cardinality constraint indicates that the attribute is functional, it is satisfied if each instance of E (resp. R) is associated to at most one value of A;

- an *is-a* constraint between an entity E_1 (or a relationship R_1) and an entity E_2 (resp. a relationship R_2) is satisfied if each instance of E_1 (resp. R_1) is also an instance of E_2 (resp. R_2).

The formal semantics of an ER schema can also be defined by encoding it in a logic based formalism called $\mathcal{DLR}$ [11, 10], which fully captures its semantics. $\mathcal{DLR}$ belongs to the family of *Description Logics*, which are class-based representation formalisms that allow one to express several kinds of relationships and constraints holding among classes. $\mathcal{DLR}$ enables one to capture the basic features of the ER model and in addition to specify several forms of constraints, as the ones described above, that cannot be expressed in the standard ER model [19, 8].

In the following example, we report a simple ER schema, and we show how to formalize it in $\mathcal{DLR}$.

Example 2.1 The schema shown in Figure 1 represents persons divided in males and females and the parent-child relationship. The cardinality constraints attached to the attributes of the entity Person indicate that all the attributes are functional and mandatory. The fact that the entity Person participates twice in the relationship CHILD (either as parent or child) is specifyed by two different ER-roles, represented by the numbers indicating their poitions. The cardinality constraint attached to the ER-role 2 specifies the mandatory and functional participation of this ER-role in the relationship CHILD.

The following set of $\mathcal{DLR}$ assertions exactly captures the ER schema in the figure.

$$
\begin{array}{rcl}
\mathsf{Person} & \sqsubseteq & (=1\,[1]\mathsf{name}) \sqcap \forall[1](\mathsf{name} \sqcap (2/2\colon \mathsf{NameString})) \sqcap \\
 & & (=1\,[1]\mathsf{ssn}) \sqcap \forall[1](\mathsf{ssn} \sqcap (2/2\colon \mathsf{SSNString})) \sqcap \\
 & & (=1\,[1]\mathsf{dob}) \sqcap \forall[1](\mathsf{dob} \sqcap (2/2\colon \mathsf{Date})) \sqcap \\
 & & (=1\,[1]\mathsf{income}) \sqcap \forall[1](\mathsf{income} \sqcap (2/2\colon \mathsf{Money})) \\
\mathsf{Person} & \equiv & \mathsf{Female} \sqcup \mathsf{Male} \\
\mathsf{Female} & \sqsubseteq & \neg \mathsf{Male} \\
\mathsf{CHILD} & \sqsubseteq & (1/2\colon \mathsf{Person}) \sqcap (2/2\colon \mathsf{Person}) \\
\mathsf{Person} & \sqsubseteq & (=1\,[2]\mathsf{CHILD})
\end{array}
$$

The first assertion specifies the cardinality and the domain of the attributes of Person. The next two assertions specify that persons are partitioned in females and males. The fourth assertion specifies the typing of the CHILD relationship. The last one specifies the cardinality constraint (functional and mandatory) attached to the participation of Person as second component of the relationship CHILD.

In the following we will refer to the possibility of defining views and queries over a global schema expressed in the ER model. We call them *queries over the conceptual level.* We restrict our analysis to the class of *union of conjunctive queries* (UCQs) [2]. A UCQ is a formula of the form

$$q(\vec{\mathbf{x}}) \leftarrow \mathit{conj}_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee \mathit{conj}_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$$

where

- q belongs to a new alphabet $\mathcal{Q}$ (the alphabet of queries, that is disjoint from both Γ and $\mathcal{A}$). $q(\vec{\mathbf{x}})$ is called the *head* of the query;
- the number of variables of $\vec{\mathbf{x}}$ is called the *arity* of q, and is the arity of the relation denoted by the query q (we us $\vec{\mathbf{x}}$ to denote a tuple of variables $x_1, \ldots, x_n$, for some n).
- $\mathit{conj}_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee \mathit{body}_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$ is called the *body* of the query;
- for each i, the *disjunct* $\mathit{conj}_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i)$ is a conjunction of atoms involving constants from Γ and variables $\vec{\mathbf{x}} = X_1, \ldots, X_n$ and $\vec{\mathbf{y}}_i = Y_{i,1}, \ldots, Y_{i,n_i}$ from an alphabet of variables,
- The predicates in the atoms are concepts of the conceptual schema, i.e., its entities, relationships and attributes:
  - Each entity E in $\mathcal{G}$ has an associated predicate E of arity 1. Intuitively, $E(c)$ asserts that c is an instance of entity E.
  - Each attribute A for an entity E has an associated predicate A of arity 2. Intuitively, $A(c, d)$ asserts that c is an instance of entity E and d is the value of attribute A associated to c.
  - Each relationship R among the entities $E_1, \ldots, E_n$ has an associated predicate R of arity n.
  - Each attribute A for a relationship R among the entities $E_1, \ldots, E_n$ has an associated predicate A of arity $n+1$. Intuitively, $A(c_1, \ldots, c_n, d)$ asserts that $(c_1, \ldots, c_n)$ is an instance of relationship R and d is the value of attribute A associated to $(c_1, \ldots, c_n)$.

In the integration framework that will be presented in Section 3, we also will consider views that are unions of conjunctive queries expressed over relational sources, in addition to the ones expressed over the global conceptual schema. In this case we will make use of *relational queries*, whose definition is as the one presented above, with the only difference that the predicate symbols of the atoms in the queries are relation symbols belonging to the schemas representing the sources, rather than concepts of a conceptual schema.

Finally we define the semantics of queries: given a database $\mathcal{DB}$, either a database for a relational schema or for an ER schema, the *answer of a query* q of arity n over $\mathcal{DB}$, denoted $q^{\mathcal{DB}}$, is the set of n-tuples of constants $(c_1, \ldots, c_n)$, such that, when substituting each c_i for x_i, the formula $\exists \vec{\mathbf{y}}_1 . \mathit{conj}_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee \exists \vec{\mathbf{y}}_m . \mathit{conj}_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$ evaluates to true in $\mathcal{DB}$.

Suitable inference techniques allow for carrying out the following reasoning services on queries over the conceptual level by taking into account the constraints expressed in the conceptual schema [11]:

- *Query containment.* Given two queries q_1 and q_2 (of the same arity n) over the conceptual level, we say that q_1 is *contained in* q_2, if the set of tuples denoted by q_1 is contained in the set of tuples denoted by q_2 in every database satisfying the conceptual schema.
- *Query consistency.* A query q over the conceptual level is *consistent*, if there exists a database satisfying the conceptual schema in which the set of tuples denoted by q is not empty.

- *Query disjointness.* Two queries q_1 and q_2 (of the same arity) over the conceptual level are *disjoint*, if the intersection of the set of tuples denoted by q_1 and the set of tuples denoted by q_2 is empty, in every database satisfying the conceptual schema.

3 A Framework for Data Integration

In this section we set up a formal framework for data integration. In particular, we describe the main components of a data integration system, namely, the global schema, the sources, and the mapping between the two. Finally, we provide the semantics both of the system, and of query answering.

Formally, A *data integration system* $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is the global schema, $\mathcal{S}$ is the source schema, and $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$. We describe the characteristics of the various components of a data integration system in our approach:

- The *global schema* $\mathcal{G}$ is expressed in terms of the ER model described in the previous section. We denote with $\mathcal{A}^{\mathcal{G}}$ the alphabet of the conceptual symbols used to specify the global schema.

- The *source schema* $\mathcal{S}$ is constituted by the schemas of the sources. We assume that the sources are expressed as relational tables. This is not a strong limitation, since, in case of sources of different type, we can assume that suitable wrappers present the data at the sources in relational form. Each source is modeled by a set of relations and we denote with $\mathcal{A}^{\mathcal{S}}$ the alphabet of the relational symbols of the sources. Finally, we assume that no integrity constraint is allowed over the sources.

- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$. The mapping indicates the relationships holding among concepts of the global schema and relations of the sources. We consider two approaches to specify the mapping:

  1. The *local-as-view* (LAV) approach, in which the meaning of the sources is expressed in terms of elements of the global schema. More precisely, the mapping in LAV is given by associating to each relation R in the sources a *view* V_R, i.e., a query, over the global schema. The intended meaning of such an association is that the view V_R represents the best way to characterize the tuples of R using the concepts in $\mathcal{G}$. We assume that the language used to express views in LAV is union of conjunctive queries.
  2. The *global-as-view* (GAV) approach, in which the elements of the global schema are described in terms of the sources. More precisely, the mapping in GAV is given by associating to each concept C in the global schema a *view* V_C over the sources. The intended meaning of such an association is that the view V_C represents the best way to characterize the instances of C using the relations in $\mathcal{S}$. No limitation is posed on the language used to specify the mapping in GAV.

  Note that, whereas we specify the LAV mapping in terms of queries over the conceptual model whose predicate symbols in the atoms belong to $\mathcal{A}^{\mathcal{G}}$, the GAV mapping is specified by means of relational queries expressed in terms of $\mathcal{A}^{\mathcal{S}}$.

In order to specify the semantics of a data integration system, we have to characterize, given the set of tuples satisfying the various source relations, which are the data satisfying the global schema. Hence, given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we start by considering a *source database* for $\mathcal{I}$, i.e., a database $\mathcal{D}$ for the source schema $\mathcal{S}$. Based on $\mathcal{D}$, we now 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* for $\mathcal{I}$ with respect to $\mathcal{D}$ if:

- $\mathcal{B}$ is coherent with $\mathcal{G}$;

- $\mathcal{B}$ satisfies the mapping $\mathcal{M}$, that is its instances respect the relationships defined between the global concepts and the source relations. In the case that the mapping is LAV, this means that for each view V_R associated to a source relation R, the set of tuples $V_R^{\mathcal{B}}$ that satisfies the view wrt $\mathcal{B}$ is coherent with set of tuples $R^{\mathcal{D}}$ that $\mathcal{D}$ assign to R, i.e., $R^{\mathcal{D}} \subseteq V_R^{\mathcal{B}}$. On the contrary, in the case that the mapping is GAV, satisfying it means that for each view V_C associated to a global concept C, the set of instances $C^{\mathcal{B}}$ that $\mathcal{B}$ assigns to C is coherent with set of tuples satisfying view V_C wrt $\mathcal{D}$, i.e., $V_C^{\mathcal{D}} \subseteq C^{\mathcal{B}}$.

The above definition implies that views are considered *sound*, i.e., that sources store only, but not necessary all, the tuples that satisfy the global schema. Another possibility would be to consider them *exact*. When views are exact, the mapping between the global schema and the sources is defined in such a way that, for every source database $\mathcal{D}$, in the LAV approach it holds that $R^{\mathcal{D}} = V_R^{\mathcal{B}}$ for every relation R of $\mathcal{S}$, whereas in the GAV approach it holds that $V_C^{\mathcal{D}} = C^{\mathcal{B}}$ for every concept C of $\mathcal{G}$. Finally, a third way on how to interpret the data stored at the sources with respect to the data that satisfies the global schema, is to consider the views *complete*. The fact that the views are complete implies that the sources store all the tuples that satisfy the global schema, i.e., for every source database $\mathcal{D}$, in the LAV approach it holds that $R^{\mathcal{D}} \supseteq V_R^{\mathcal{B}}$ for every relation R of $\mathcal{S}$, whereas in the GAV approach it holds that $V_C^{\mathcal{D}} \supseteq C^{\mathcal{B}}$ for every concept C of $\mathcal{G}$. In the following we only refer to sound views.

Given a source database $\mathcal{D}$ for $\mathcal{I}$, the semantics of $\mathcal{I}$ wrt $\mathcal{D}$, denoted $sem(\mathcal{I}, \mathcal{D})$, is defined as follows:

$$sem(\mathcal{I}, \mathcal{D}) = \{ \mathcal{B} \mid \mathcal{B} \text{ is a legal global database for } \mathcal{I} \text{ wrt } \mathcal{D} \}$$

From the above definitions, it is easy to see that given a legal source database $\mathcal{D}$, in general several global coherent databases exist. In specific cases, no global coherent database may exist, for example if the set of constraints of the global schema is inconsistent, or exactly one coherent database may exist.

We conclude the section by defining the notion of query posed to the data integration system. A query Q to a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ is a query over the conceptual level, whose atoms have symbols in $\mathcal{G}$ as predicates, as illustrated in Section 2. Our goal here is to specify which are the tuples that form the answer to a query posed to a data integration system $\mathcal{I}$. The fact that, given a source database $\mathcal{D}$, several global databases may exist that are legal for $\mathcal{I}$ with respect to $\mathcal{D}$ complicates this task. In order to address this problem, we follow a first-order logic approach: a tuple $(c_1, \ldots, c_n)$ is considered an answer to the query only if it is a *certain* answer, i.e., it satisfies the query in *every* database that belongs to the semantics of the data integration system. Formally, the *set of certain answers* $Q^{\mathcal{I},\mathcal{D}}$ to Q with respect to $\mathcal{I}$ and $\mathcal{D}$ is the set of tuples $(c_1, \ldots, c_n)$ such that $(c_1, \ldots, c_n) \in Q^{\mathcal{B}}$, for each $\mathcal{B} \in sem(\mathcal{I}, \mathcal{D})$.

Notice that, in a data integration system $\mathcal{I}$, either a LAV or a GAV system, answering queries is essentially an extended form of reasoning in the presence of incomplete information [54]. Indeed, when we answer the query, we know only the extensions of the sources, and this provides us with only partial information on the global database. Moreover, since the query language may admit various forms of incomplete information (due to union, for instance), there are in general several possible databases coherent with $\mathcal{I}$.

4 Reconciliation in LAV

When the mapping between the global schema and the sources is specified following the local-as-view approach, a query over the conceptual level is associated to each source relation. In this section we first summarize the framework at the basis of data integration in the LAV setting, and we describe how to specify the integration system at the logical level. We address the problem of data reconciliation by specifying suitable Reconciliation Correspondences, and then present a method for the synthesis of mediators that takes such correspondences into account.

4.1 LAV Framework

In the LAV approach we explicitly model the data managed by the integration system at different levels of abstraction [12, 14, 13]. We refer to the abstract framework introduced in Section 3.

- At the *conceptual level*, the global schema contains a conceptual representation of the data managed by the integration system, including a conceptual representation of the data residing in sources, and of the global concepts and relationships that are of interest for the application. As detailed in the previous section, the global schema is expressed in terms of the ER model. In the rest of the section, we will use the term "conceptual level" instead of "global schema" to emphasize that the global schema contains a representation of the reality of interest at the conceptual level.

- At the *logical level*, the source schema provides a representation of the sources in terms of a logical data model, in our case the relational model. If the integration system maintains also materialized integrated data (e.g., in the case of a Data Warehouse), the logical level contains also a representation of such data in terms of the logical model; we call such a representation the *materialized view schema*.

The relationship between the conceptual level and the logical level, is represented explicitly by specifying a *mapping* between corresponding objects of the two levels. More precisely, in the *local-as-view approach* the link between the logical representation and the conceptual representation of the sources is formally defined by associating each source relation with a query that describes its content in terms of a query over the conceptual level (as defined in Section 2). In other words, the logical content of a source relation is described in terms of a view over the virtual database represented by the conceptual level.

We remind that we assume that physical structures are mapped to logical structures by means of suitable wrappers, which encapsulate the sources. The wrapper hides how the source actually stores its data, the data model it adopts, etc., and presents the source as a set of relations. In particular, we assume that all attributes in the relations are of interest to the information integration application (attributes that are not of interest are hidden by the wrapper). Relation attributes are thus modeled as either entity attributes or relationship attributes in the conceptual level.

The materialized view schema (if present at all), which expresses the logical content of the materialized views, is provided in terms of a set of relations. Similarly to the case of the sources, each relation of the materialized view schema is described in terms of a query over the conceptual level.

In the following, we will also consider queries whose body may contain special predicates that do not appear in the conceptual level.

4.2 Source and Materialized Views Logical Schema Descriptions

As mentioned above, we express the relational tables constituting both the source schema and the materialized views schema in terms of queries over the conceptual level. Such queries are a powerful tool for modeling the logical level of the sources and the materialized views. To take into account that at the conceptual level we deal with (conceptual) objects, while at the logical level the relations store values (rather than conceptual objects), queries over the conceptual level have the following characteristics:

- Relational tables are composed of tuples of values, which are the only kind of objects at the logical level. Therefore, each variable in the head of the query represents a value (not a conceptual object).

- Each variable appearing in the body of the query either denotes a conceptual object or a value, depending on the atoms in which it appears. Since, in each database that satisfies the conceptual level, conceptual objects and values are disjoint sets, no query can contain a variable which can be instantiated by both a conceptual object and a value.

- Each conceptual object is represented at the logical level by a tuple of values. Thus, a mechanism is needed to express this kind of correspondence between a tuple of values and the conceptual object it represents. This is taken into account by the notion of *adornment* introduced below.

4.2.1 Source Schema Description

The query associated with a source relation provides the glue between the conceptual and the logical representation. However, to capture in a precise way the data in the sources, more information is needed in order to describe the actual structure of the data in the relation. This is done by the *adornment* associated to the relation, whose role is to declare the domains of the columns of the table, and which are the attributes of the table that are used to identify the objects of the conceptual level. In other words, the adornment is used to make explicit how the objects of the conceptual representation are coded into values of the logical representation.

An *adorned query* is an expression of the form

$$Q(\vec{\mathbf{x}}) \ \leftarrow \ \mathit{body}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ \mid \ \alpha_1, \ldots, \alpha_n$$

where $Q(\vec{\mathbf{x}}) \leftarrow \mathit{body}(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ is a query over the conceptual level and $\alpha_1, \ldots, \alpha_n$ constitutes the *adornment* in which each α_i is an *annotation* on variables appearing in $\vec{\mathbf{x}}$. In particular:

- For each $X \in \vec{\mathbf{x}}$, we have an annotation of the form

  $$X :: V$$

  where V is a domain expression. Such an annotation is used to specify how values bound to X are represented in the table at the logical level.

- For each tuple of variables $\vec{\mathbf{z}} \subseteq \vec{\mathbf{x}}$ that is used for identifying in Q a conceptual object $Y \in \vec{\mathbf{y}}$ mentioned in $q(\vec{\mathbf{x}}, \vec{\mathbf{y}})$, we have an annotation of the form

  $$\mathit{Identify}([\vec{\mathbf{z}}], Y)$$

  where we have grouped the variables $\vec{\mathbf{z}}$ into a single argument $[\vec{\mathbf{z}}]$. Such an annotation makes it explicit that the tuple of values $\vec{\mathbf{z}}$ is a representation of the conceptual object Y.

We illustrate by means of an example how adorned queries are used to specify the information content of source tables. We use $X_1, \ldots, X_k :: V$ as an abbreviation for $X_1 :: V, \ldots, X_k :: V$.

Example 4.1 Suppose we store in two sources $\mathcal{S}_1$ and $\mathcal{S}_2$ parent-child relationships, according to the conceptual level shown in Figure 1. Source $\mathcal{S}_1$ contains the information about fathers and their children, in terms of a relational table FATHER_1 which stores all such pairs. Similarly, source $\mathcal{S}_2$ contains the information about mothers and their children in terms of a relational table MOTHER_2. We assume that in source $\mathcal{S}_1$ persons (both fathers and children) are identified by their name and date of birth, while in source $\mathcal{S}_2$ persons are identified by their social security number. Hence, we can specify the information content of the two tables by the following adorned queries:

$$\begin{array}{lcl}
\mathsf{FATHER}_1(N_f, D_f, N_c, D_c) & \leftarrow & \mathsf{Male}(F) \wedge \mathsf{Person}(C) \wedge \mathsf{CHILD}(F, C) \wedge \\
& & \mathsf{name}(F, N_f) \wedge \mathsf{dob}(F, D_f) \wedge \mathsf{name}(C, N_c) \wedge \mathsf{dob}(C, D_c) \\
& & \mid\ N_f, N_c :: \mathsf{NameString}, \\
& & \quad D_f, D_c :: \mathsf{Date}, \\
& & \quad \mathit{Identify}([N_f, D_f], F),\ \mathit{Identify}([N_c, D_c], C) \\
\mathsf{MOTHER}_2(S_m, S_c) & \leftarrow & \mathsf{Female}(M) \wedge \mathsf{Person}(C) \wedge \mathsf{CHILD}(M, C) \wedge \\
& & \mathsf{ssn}(M, S_m) \wedge \mathsf{ssn}(C, S_c) \\
& & \mid\ S_m, S_c :: \mathsf{SSNString}, \\
& & \quad \mathit{Identify}([S_m], M),\ \mathit{Identify}([S_c], C)
\end{array}$$

■

Example 4.2 Referring again to the conceptual level shown in Figure 1, we want to model two sources storing the information about the income of persons. Source $\mathcal{S}_1$ stores the income per month of males in a table INCOME_1, while source $\mathcal{S}_2$ stores the income per year of females in a table INCOME_2, and the content of the tables is specified by the following adorned queries:

$$\begin{array}{lcl}
\mathsf{INCOME}_1(S_m, I) & \leftarrow & \mathsf{Male}(M) \wedge \mathsf{ssn}(M, S_m) \wedge \mathsf{income}(M, I) \\
& & \mid\ S_m :: \mathsf{SSNString}, \\
& & \quad I :: \mathsf{IncomePerMonth}, \\
& & \quad \mathit{Identify}([S_m], M) \\
\mathsf{INCOME}_2(S_f, I) & \leftarrow & \mathsf{Female}(F) \wedge \mathsf{ssn}(F, S_f) \wedge \mathsf{income}(F, I) \\
& & \mid\ S_f :: \mathsf{SSNString}, \\
& & \quad I :: \mathsf{IncomePerYear}, \\
& & \quad \mathit{Identify}([S_f], F)
\end{array}$$

■

The adorned query associated to a table in a source contains a lot of information that can be profitably used in analyzing the quality of the integration system design process. Indeed, the adorned query precisely formalizes the content of a source table in terms of a query over the conceptual level, the domains of each attribute of the table, and the attributes used to identify entities at the conceptual level. One important check that we can carry out over the logical specification of a source is whether the adorned query associated with a table in a source is consistent or not. Let Q be an adorned query and let B be its body. The query B is said to be *inconsistent* with respect to the conceptual level $\mathcal{G}$, if for every database $\mathcal{DB}$ coherent with $\mathcal{G}$, the evaluation of B with respect to $\mathcal{DB}$ is empty. An adorned query Q may be inconsistent with respect to the conceptual level $\mathcal{G}$ because the body B of Q is inconsistent with respect to $\mathcal{G}$. Inference techniques allow us to check for these forms of inconsistency [14]. Another reason for inconsistency in specifying a table may be due to annotations that are incoherent with respect to what specified in $\mathcal{G}$. In particular, domain expressions used in the adornment can be inconsistent with respect to the constraints on domains specified at the conceptual level. Standard propositional reasoning tools [5, 57, 23] can be adopted to detect such forms of inconsistency.

4.2.2 Materialized Views Schema Description

Similarly to the case of source relations, the views to be materialized are described as adorned queries over the conceptual level.

Note that the adorned query associated to a table in a source is the result of a reverse engineering analysis of the source, whereas in this case the adorned query is a high-level specification of what we want to materialize in views, and thus of the mediator for loading such a materialized view. Since we express the semantics of the tables to materialize in terms of the conceptual level, also the materialized views are seen as views of such a conceptual level.

Example 4.1 (cont.) Suppose we want materialize (e.g., in a Data Warehouse) all pairs of male-female with a child in common. The pairs have to be stored in a table $\mathsf{COMMONCHILD_{DW}}$, identifying each person by its social security number. We can specify the content of the table wrt the conceptual level by the following adorned query:

$$\begin{array}{rcl} \mathsf{COMMONCHILD_{DW}}(S_f, S_m) & \leftarrow & \mathsf{Male}(F) \wedge \mathsf{ssn}(F, S_f) \wedge \mathsf{CHILD}(F, C) \wedge \\ & & \mathsf{Female}(M) \wedge \mathsf{ssn}(M, S_m) \wedge \mathsf{CHILD}(M, C) \\ & & \mid\ S_f, S_m :: \mathsf{SSNString}, \\ & & \quad \mathit{Identify}([S_f], F),\ \mathit{Identify}([S_m], M) \end{array}$$ ∎

Example 4.2 (cont.) Suppose we want to store in the materialized views pairs of persons with the same income. The pairs, together with the common income per year, have to be stored in a table $\mathsf{SAMEINCOME_{DW}}$, identifying each person by its social security number. We can specify the content of the table wrt the conceptual level by the following adorned query:

$$\begin{array}{rcl} \mathsf{SAMEINCOME_{DW}}(S_1, S_2, I) & \leftarrow & \mathsf{Person}(P_1) \wedge \mathsf{ssn}(P_1, S_1) \wedge \mathsf{income}(P_1, I) \wedge \\ & & \mathsf{Person}(P_2) \wedge \mathsf{ssn}(P_2, S_2) \wedge \mathsf{income}(P_2, I) \\ & & \mid\ S_1, S_2 :: \mathsf{SSNString}, \\ & & \quad I :: \mathsf{IncomePerYear}, \\ & & \quad \mathit{Identify}([S_1], P_1),\ \mathit{Identify}([S_2], P_2) \end{array}$$ ∎

The reasoning services provided at the logical level make it possible to automatically generate the correct mediators for the loading of the materialized views. As illustrated in Section 4.4, this is realized by means of a query rewriting technique which uses query containment as its basic reasoning service.

4.2.3 Schematic Differences

The mechanism used in our framework for specifying adorned queries is able to cope with *schematic differences* [51]. The example below illustrates a case where there are various schematic differences, both among the sources, and between the sources and the conceptual level.

Example 4.3 Suppose that the conceptual level contains an entity Service with three attributes, date, type, and price, specifying respectively the date, the type, and the cost of the service. Suppose that source $\mathcal{S}_1$ represents information only on services of type t_1 and t_2, by means of two relations: t1 and t2, where $\mathsf{t1}(D, P)$ means that service t_1 costs P Italian Lira at date D, and $\mathsf{t2}(D, P)$ means that service t_2 costs P Italian Lira at date D. Suppose that source $\mathcal{S}_2$ represents information only on services t_3 and t_4 by means of a relation Serv_2, where $\mathsf{Serv}_{3,4}(D, P_3, P_4)$ means that services t_3 and t_4 cost P_3 and P_4 Euro respectively at date D. Finally, suppose that source $\mathcal{S}_3$ represents the information only for a certain date d by means of another relation Serv_d. The various relations in the three sources can be specified by means of the following adorned queries:

$$\begin{array}{rcl} \mathsf{t1}(D, P) & \leftarrow & \mathsf{Service}(S) \wedge \mathsf{date}(S, D) \wedge \mathsf{type}(S, \text{`}\mathtt{t1}\text{'}) \wedge \mathsf{price}(S, P) \\ & & \mid\ D :: \mathsf{Date},\ P :: \mathsf{ItalianLira},\ \mathit{Identify}([\text{`}\mathtt{t1}\text{'}, D], S) \end{array}$$

$$
\begin{array}{rcl}
\mathsf{t2}(D,P) & \leftarrow & \mathsf{Service}(S) \land \mathsf{date}(S,D) \land \mathsf{type}(S,\texttt{'t2'}) \land \mathsf{price}(S,P) \\
& & \mid\ D :: \mathsf{Date},\ P :: \mathsf{ItalianLira},\ \mathit{Identify}([\texttt{'t2'},D],S) \\
\mathsf{Serv}_{3,4}(D,P_3,P_4) & \leftarrow & \mathsf{Service}(S_3) \land \mathsf{date}(S_3,D) \land \mathsf{type}(S_3,\texttt{'t3'}) \land \mathsf{price}(S_3,P_3) \land \\
& & \mathsf{Service}(S_4) \land \mathsf{date}(S_4,D) \land \mathsf{type}(S_4,\texttt{'t4'}) \land \mathsf{price}(S_4,P_4) \\
& & \mid\ D :: \mathsf{Date},\ P_3 :: \mathsf{Euro},\ P_4 :: \mathsf{Euro}, \\
& & \quad \mathit{Identify}([\texttt{'t3'},D],S_3),\ \mathit{Identify}([\texttt{'t4'},D],S_4) \\
\mathsf{Serv}_d(T,P) & \leftarrow & \mathsf{Service}(S) \land \mathsf{date}(S,\texttt{'d'}) \land \mathsf{type}(S,T) \land \mathsf{price}(S,P) \\
& & \mid\ T :: \mathsf{TypeString},\ P :: \mathsf{Euro},\ \mathit{Identify}([T,\texttt{'d'}],S)
\end{array}
$$

■

4.3 Reconciliation Correspondences

Assume that the integration system requires to compute a query, and that an adorned query Q specifies, in terms of the conceptual level, how the data to compute the query has to be obtained. This can be either because the query Q is required to compute the answer to a user query posed to the integration system, or because Q results from the decision on which data to materialize in the integration system. One crucial task is to design the *mediator* for Q, i.e., the program that accesses the sources to obtain the correct data for computing Q. Designing the mediator for Q requires first of all to reformulate the query associated with Q in terms of the source relations. However, such a reformulation is not sufficient. The task of mediator design is complicated by the possible heterogeneity between the data in the sources. We have already mentioned the most important ones, namely, mismatches between data referring to the same real world object, errors in the data stored in the sources, inconsistencies between values representing the properties of the real world objects in different sources.

Our proposal to cope with this problem is based on the notion of Reconciliation Correspondence. Indeed, in order to anticipate possible errors, mismatches and inconsistencies between data in the sources, our approach allows the designer to declaratively specify the correspondences between data in different schemas (either source schemas or materialized views schema). Such specification is done through special assertions, called *Reconciliation Correspondences*.

Reconciliation Correspondences are defined in terms of relations, similarly to the case of the relations describing the sources and the materialized views at the logical level. The difference with source and materialized views relations is that we conceive Reconciliation Correspondences as non-materialized relations, in the sense that their extension is computed by an associated program whenever it is needed. In particular, each Reconciliation Correspondence is specified as an adorned query with an associated *program* that is called to compute the extension of the virtual relation. Note that we do not consider the actual code of the program but just its input and output parameters.

We distinguish among three types of correspondences, namely Conversion, Matching, and Merging Correspondences.

4.3.1 Conversion Correspondences

Conversion Correspondences are used to specify that data in one table can be converted into data of a different table, and how this conversion is performed. They are used to anticipate various types of data conflicts that may occur in loading data coming from different sources into the materialized views.

Formally, a *Conversion Correspondence* has the following form:

$$
\begin{array}{rcl}
\mathit{convert}_C([\vec{\mathbf{x}}_1],[\vec{\mathbf{x}}_2]) & \leftarrow & \mathit{Equiv}([\vec{\mathbf{x}}_1],[\vec{\mathbf{x}}_2]) \land \mathit{conj}(\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2,\vec{\mathbf{y}}) \\
& & \mid\ \alpha_1,\ldots,\alpha_n \\
& & \texttt{through}\ \mathit{program}(\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2,\vec{\mathbf{y}})
\end{array}
$$

where: $convert_C$ is the *conversion predicate* defined by the correspondence; *Equiv* is a special predicate whose intended meaning is that $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{x}}_2$ actually represent the same data[1]; *conj* is a conjunction of atoms, which specifies the conditions under which the conversion is applicable; $\alpha_1, \ldots, \alpha_n$ is an adornment of the query; *program* denotes a program that performs the conversion. In general, the program needs to take into account the additional parameters specified in the condition to actually perform the conversion. The conversion has a direction. In particular, it operates from a tuple of values satisfying the conditions specified for $\vec{\mathbf{x}}_1$ in *conj* and in the adornment to a tuple of values satisfying the conditions specified for $\vec{\mathbf{x}}_2$. This means that the conversion program receives as input a tuple $\vec{\mathbf{x}}_1$, and returns the corresponding tuple $\vec{\mathbf{x}}_2$, possibly using the additional parameters $\vec{\mathbf{y}}$ to perform the conversion. Notice that we will have to take into account that the conversion has a direction also when we make use of the correspondence for populating the materialized views.

Example 4.1 (cont.) Since we know that persons in the materialized views are identified by their social security number, while in source $\mathcal{S}_1$ they are identified by their name and date of birth, we have to provide a mean to convert the name and date of birth of a person to his social security number. Let name_dob_to_ssn be a suitable program that performs the conversion, taking as input name and date of birth, and returning the social security number. To give a declarative account of this ability, we provide the following Conversion Correspondence:

$$\begin{array}{rcl} \mathit{convert}_{\mathit{person}}([N,D],[S]) & \leftarrow & \mathit{Equiv}([N,D],[S]) \wedge \mathsf{Person}(P) \wedge \\ & & \mathsf{name}(P,N) \wedge \mathsf{dob}(P,D) \wedge \mathsf{ssn}(P,S) \\ & & \mid\ N :: \mathsf{NameString},\ D :: \mathsf{Date},\ S :: \mathsf{SSNString}, \\ & & \quad \mathit{Identify}([N,D],P),\ \mathit{Identify}([S],P) \\ & & \mathtt{through}\ \mathsf{name\_dob\_to\_ssn}(N,D,S) \end{array}$$

The Conversion Correspondence specifies the condition under which a name N and date of birth D can be converted to a social security number S, by requiring the existence of a person with name N, date of birth D, and social security number S. Notice that the predicates in the body of the Conversion Correspondence play a fundamental role in restricting the applicability of the conversion to the proper attributes of persons. ■

In the example above we have used a correspondence that converts between different representations of the same object. Since a representation may consist of a tuple of values (e.g., $[N, D]$) this is a typical situation in which we compare tuples, possibly of different arity. Our approach requires indeed that, in order to compare tuples, we have a common underlying object, which is used to make explicit the relationship between the components of the two tuples. In the case where we do not have such an underlying object, we still allow to compare single values in different domains, as shown in the following example.

Example 4.2 (cont.) To translate incomes per month to incomes per year we define the following Conversion Correspondence:

$$\begin{array}{rcl} \mathit{convert}_{\mathit{income}}([I_m],[I_y]) & \leftarrow & \mathit{Equiv}([I_m],[I_y]) \\ & & \mid\ I_m :: \mathsf{IncomePerMonth},\ I_y :: \mathsf{IncomePerYear} \\ & & \mathtt{through}\ \mathsf{income\_month\_to\_year}(I_m,I_y) \end{array}$$ ■

[1] *Equiv* plays a special role during the construction of the rewriting for the synthesis of the mediator, as explained in Section refsec-mediator.

4.3.2 Matching Correspondences

Matching Correspondences are used to specify how data in different source tables, typically referring to different sources, can match. Formally, a *Matching Correspondence* has the following form:

$$\begin{array}{lll} match_M([\vec{\mathbf{x}}_1],\ldots,[\vec{\mathbf{x}}_k]) & \leftarrow & Equiv([\vec{\mathbf{x}}_1],[\vec{\mathbf{x}}_2]) \wedge \ldots \wedge Equiv([\vec{\mathbf{x}}_{k-1}],[\vec{\mathbf{x}}_k]) \wedge \\ & & conj(\vec{\mathbf{x}}_1,\ldots,\vec{\mathbf{x}}_k,\vec{\mathbf{y}}) \\ & & \mid\ \alpha_1,\ldots,\alpha_n \\ & & \texttt{through}\ program(\vec{\mathbf{x}}_1,\ldots,\vec{\mathbf{x}}_k,\vec{\mathbf{y}}) \end{array}$$

where $match_M$ is the *matching predicate* defined by the correspondence, *Equiv* is as before, *conj* specifies the conditions under which the matching is applicable, $\alpha_1,\ldots,\alpha_n$ is again an adornment of the query, and *program* denotes a program that performs the matching. The program receives as input k tuples of values satisfying the conditions (and possibly the additional parameters in the condition) and returns whether the tuples match or not.

Example 4.1 (cont.) To compare persons in source $\mathcal{S}_1$ with persons in source $\mathcal{S}_2$, we have to provide a mean to compare persons identified by their name and date of birth with persons identified by their social security number. Let name_dob_matches_ssn be a suitable program that, given name, date of birth, and the social security number as input, returns whether they correspond to the same person or not. To give a declarative account of this program, we provide the following Matching Correspondence:

$$\begin{array}{lll} match_{person}([N,D],[S]) & \leftarrow & Equiv([N,D],[S]) \wedge \mathsf{Person}(P) \wedge \\ & & \mathsf{name}(P,N) \wedge \mathsf{dob}(P,D) \wedge \mathsf{ssn}(P,S) \\ & & \mid\ N :: \mathsf{NameString},\ D :: \mathsf{Date},\ S :: \mathsf{SSNString}, \\ & & \quad Identify([N,D],P),\ Identify([S],P) \\ & & \texttt{through}\ \mathsf{name\_dob\_matches\_ssn}(N,D,S) \end{array}$$ ■

Notice that the Matching Correspondence $match_{person}$ and the Conversion Correspondence $convert_{person}$ look the same. However there are fundamental differences between them. The associated programs behave differently. In particular, name_dob_to_ssn(N,D,S) is used to generate the social security number S of the person with name N and date of birth D, whereas name_dob_matches_ssn(N,D,S) is used to verify that N and D, and S refer to the same person. Consequently, when the Matching Correspondence and the Conversion Correspondence are used as atoms in a query Q, they have to be used according to different binding patterns of the variables occurring in the atoms. In particular, in the Matching Correspondence all variables have to be bound, and therefore cannot occur in the head of the query Q. Instead, in the Conversion Correspondence, the variables in the first tuple need to be bound, while the ones in the second tuple are free. Hence the variables in the second tuple can occur in the head of Q.

4.3.3 Merging Correspondences

Merging Correspondences are used to assert how we can merge data in different sources into data that contributes to the answer to a query. Formally, a *Merging Correspondence* has the following form:

$$\begin{array}{lll} merge_R([\vec{\mathbf{x}}_1],\ldots,[\vec{\mathbf{x}}_k],[\vec{\mathbf{x}}_0]) & \leftarrow & Equiv([\vec{\mathbf{x}}_1],[\vec{\mathbf{x}}_0]) \wedge \ldots \wedge Equiv([\vec{\mathbf{x}}_k],[\vec{\mathbf{x}}_0]) \wedge \\ & & conj(\vec{\mathbf{x}},\ldots,\vec{\mathbf{x}}_k,\vec{\mathbf{x}}_0,\vec{\mathbf{y}}) \\ & & \mid\ \alpha_1,\ldots,\alpha_n \\ & & \texttt{through}\ program(\vec{\mathbf{x}}_1,\ldots,\vec{\mathbf{x}}_k,\vec{\mathbf{x}}_0,\vec{\mathbf{y}}) \end{array}$$

where $merge_R$ is the *merging predicate* defined by the correspondence, *Equiv* is as before, *conj* specifies the conditions under which the merging is applicable, and *program* is a program that performs the merging. Such correspondence specifies that the k tuples of values $\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_k$ coming from the sources are merged into the tuple $\vec{\mathbf{x}}_0$. Therefore, the associated program receives as input k tuples of values (and possibly the additional parameters in the condition) and returns a tuple which is the result of the merging. Example 4.5 below illustrates the use of a Merging Correspondence.

4.3.4 Methodological Guidelines

The task of specifying suitable Reconciliation Correspondences is a responsibility of the designer. Once such Reconciliation Correspondences are specified, they are profitably exploited to automatically generate mediators, as described in Section 4.4. In the task of specifying Reconciliation Correspondences, the system can assist the designer in various ways.

First of all, since each Reconciliation Correspondence is declaratively specified as an adorned query, all reasoning tasks available for such queries can be exploited to check desirable properties of the correspondence. In particular, the system can check the consistency of queries, rejecting inconsistent ones and giving the designer useful feedback. Also, the system can automatically detect whether the adorned queries associated with different correspondences are contained in each other (or are equivalent). This is an indication for redundancy in the specification. However, to determine whether a correspondence is actually redundant, one has to consider also the types of the correspondences and the programs associated with them. E.g., a less general query, thus specifying stricter conditions for applicability, may still be useful in the case where the associated program takes advantage of the specialization and operates more efficiently.

In practice, the system automatically asserts several correspondences *by default*, thus simplifying the task of the designer.

- Several of the Reconciliation Correspondences that must be specified will have a very simple form, since they will correspond simply to equality. In particular, for each domain D in the conceptual level, the following Reconciliation Correspondences are asserted by default:

$$\begin{array}{rcl}
\mathit{convert}_D([X],[Y]) & \leftarrow & \mathit{Equiv}([X],[Y]) \\
& & \mid\ X, Y :: D \\
& & \texttt{through}\ \mathsf{identity}(X,Y) \\
\mathit{match}_D([X],[Y]) & \leftarrow & \mathit{Equiv}([X],[Y]) \\
& & \mid\ X, Y :: D \\
& & \texttt{through}\ \mathsf{none} \\
\mathit{merge}_D([X],[Y],[Z]) & \leftarrow & \mathit{Equiv}([X],[Z]) \land \mathit{Equiv}([Y],[Z]) \\
& & \mid\ X, Y, Z :: D \\
& & \texttt{through}\ \mathsf{identity}(X,Y,Z)
\end{array}$$

  where identity is the program that computes the identity function for values of domain D, and the matching correspondence has no associated program. When the designer provides an Reconciliation Correspondence referring to a domain D, then the automatic generation of the default correspondences for D is inhibited.

- Similarly, for each annotation of the form $\mathit{Identify}([X_1, \ldots, X_k], X)$ appearing in the adornment of a query q defining a source or materialized views table Q, and such that $X_1 :: D_1, \ldots, X_k :: D_k$ are the annotations in q specifying the domains associated with $X_1, \ldots, X_k$, the following Reconciliation Correspondences are asserted by default:

$$\begin{array}{rcl}
\mathit{convert}_{D_1,\ldots,D_k}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]) & \leftarrow & \mathit{Equiv}([X_1],[Y_1]) \land \ldots \land \mathit{Equiv}([X_k],[Y_k]) \\
& & \mid\ X_1, Y_1 :: D_1,\ \ldots,\ X_k, Y_k :: D_k \\
& & \texttt{through}\ \mathsf{identity}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}])
\end{array}$$

$$
\begin{aligned}
\mathit{match}_{D_1,\ldots,D_k}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]) \;\leftarrow\;& \mathit{Equiv}([X_1],[Y_1]) \wedge \ldots \wedge \mathit{Equiv}([X_k],[Y_k]) \\
& \mid\; X_1, Y_1 :: D_1,\; \ldots,\; X_k, Y_k :: D_k \\
& \texttt{through}\ \mathsf{none} \\
\mathit{merge}_{D_1,\ldots,D_k}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}],[\vec{\mathbf{z}}]) \;\leftarrow\;& \mathit{Equiv}([X_1],[Z_1]) \wedge \mathit{Equiv}([Y_1],[Z_1]) \wedge \ldots \wedge \\
& \mathit{Equiv}([X_k],[Y_k]) \wedge \mathit{Equiv}([Y_k],[Z_k]) \\
& \mid\; X_1, Y_1, Z_1 :: D_1,\; \ldots,\; X_k, Y_k, Z_k :: D_k \\
& \texttt{through}\ \mathsf{identity}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}],[\vec{\mathbf{z}}])
\end{aligned}
$$

where $\vec{\mathbf{x}}$ abbreviates $X_1, \ldots, X_k$, $\vec{\mathbf{y}}$ abbreviates $Y_1, \ldots, Y_k$, and $\vec{\mathbf{z}}$ abbreviates $Z_1, \ldots, Z_k$.

- For each Conversion Correspondence $\mathit{convert}_i$ asserted by the designer, the system automatically asserts the Matching Correspondence

$$
\begin{aligned}
\mathit{match}_i([\vec{\mathbf{x}}_1],[\vec{\mathbf{x}}_2]) \;\leftarrow\;& \mathit{convert}_i([\vec{\mathbf{x}}_1],[\vec{\mathbf{y}}]) \wedge \mathit{Equiv}(\vec{\mathbf{x}}_2,\vec{\mathbf{y}}) \\
& \texttt{through}\ \mathsf{identity}(\vec{\mathbf{x}}_2,\vec{\mathbf{y}})
\end{aligned}
$$

  Observe that a new tuple $\vec{\mathbf{y}}$ in the body of the correspondence is necessary to respect the binding pattern for $\mathit{convert}_i$. The program associated with $\mathit{convert}_i$ instantiates $\vec{\mathbf{y}}$, and identity compares the obtained value with $\vec{\mathbf{x}}_2$.

- For each Conversion Correspondence $\mathit{convert}_i$ asserted by the designer and for each Matching Correspondence match_j asserted by the designer or by default, the system automatically asserts the Merging Correspondence

$$
\begin{aligned}
\mathit{merge}_{i,j}([\vec{\mathbf{x}}_1],[\vec{\mathbf{x}}_2],[\vec{\mathbf{x}}_0]) \;\leftarrow\;& \mathit{match}_i([\vec{\mathbf{x}}_1],[\vec{\mathbf{x}}_2]) \wedge \mathit{convert}_j([\vec{\mathbf{x}}_1],[\vec{\mathbf{x}}_0]) \\
& \texttt{through}\ \mathsf{none}
\end{aligned}
$$

Example 4.1 (cont.) For the domain SSNString the system automatically asserts, e.g., the Conversion Correspondence

$$
\begin{aligned}
\mathit{convert}_{\mathsf{SSNString}}([S_1],[S_2]) \;\leftarrow\;& \mathit{Equiv}([S_1],[S_2]) \\
& \mid\; S_1, S_2 :: \mathsf{SSNString} \\
& \texttt{through}\ \mathsf{identity}(S_1, S_2)
\end{aligned}
$$

Similarly, since the adornment of the query defining the table FATHER_1 contains the annotations $\mathit{Identify}([N_f, D_f], F)$, N_f :: NameString, and D_f :: Date, the system automatically asserts, e.g., the Matching Correspondence

$$
\begin{aligned}
\mathit{match}_{\mathsf{NameString},\mathsf{Date}}([N_1, D_1],[N_2, D_2]) \;\leftarrow\;& \mathit{Equiv}([N_1],[N_2]) \wedge \mathit{Equiv}([D_1],[D_2]) \\
& \mid\; N_1, N_2 :: \mathsf{NameString},\; D_1, D_2 :: \mathsf{Date} \\
& \texttt{through}\ \mathsf{none}
\end{aligned}
$$

■

In addition, the designer may use already specified Reconciliation Correspondences to define new ones.

Example 4.4 The designer may want to define a Matching Correspondence between two tuples by using two already defined Conversion Correspondences, which convert to a common representation, and then by comparing the converted values. In this case, she could provide the following definition of the Matching Correspondence:

$$
\begin{aligned}
\mathit{match}_{X,Y}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]) \;\leftarrow\;& \mathit{convert}_{X,Z}([\vec{\mathbf{x}}],[\vec{\mathbf{z}}_1]) \wedge \mathit{convert}_{Y,Z}([\vec{\mathbf{y}}],[\vec{\mathbf{z}}_2]) \wedge \mathit{Equiv}([\vec{\mathbf{z}}_1],[\vec{\mathbf{z}}_2]) \\
& \texttt{through}\ \mathsf{identity}([\vec{\mathbf{z}}_1],[\vec{\mathbf{z}}_2])
\end{aligned}
$$

Observe that, in this case, the program associated with the Matching Correspondence is used only to check whether the converted values are identical. ■

Similarly, the designer could define a Merging Correspondence by reusing appropriate Conversion or Matching Correspondences that exploit a common representation, as shown in the following example.

Example 4.5 Suppose we want to merge prices in Italian Lira and Deutsche Mark into prices in US Dollars, and we have programs that allowed us to define the Conversion Correspondences $convert_{L,E}$ from Italian Lira to Euro, $convert_{M,E}$ from Deutsche Mark to Euro, and $convert_{E,D}$ from Euro to US Dollar. Then we can obtain the desired Merging Correspondence as follows:

$$\begin{array}{lcl} merge_{L,M,D}([L],[M],[D]) & \leftarrow & convert_{L,E}([L],[E_1]) \wedge convert_{M,E}([M],[E_2]) \wedge \\ & & Equiv([E_1],[E_2]) \wedge convert_{E,D}([E_1],[D]) \\ & & \texttt{through}\ \mathsf{identity}([E_1],[E_2]) \end{array}$$ ■

4.4 Specification of Mediators

As we said before, our goal is to provide support for the design of the mediator for Q, i.e., the program that accesses the sources and loads the correct data into the relation Q. In general, the design of mediators requires a sophisticated analysis of the data, which aims at specifying, for every relation to be computed, how the tuples of the relation should be constructed from a suitable set of tuples extracted from the sources. Mediator design is typically performed by hand and is a costly step in the overall design process of an integration system. The framework presented here is also based on a detailed analysis of the data and of the information needs. However, the knowledge acquired by such an analysis is explicitly expressed in the description of source and materialized views relations, and in the Reconciliation Correspondences. Hence, such a knowledge can be profitably exploited to support the design of the mediators associated to relations.

Suppose we have decided to compute a new relation Q, and let q be the adorned query associated to Q. Our technique requires to proceed as follows.

1. We determine how the data in Q can be obtained from the data stored in already defined materialized views (if present), the data stored in source relations, and the data returned by the programs that perform conversion, matching, and merging associated to the Reconciliation Correspondences. This is done by looking for a *rewriting* of q in terms of the available adorned queries, i.e., a new query q' contained in q whose atoms refer to (i) the already available materialized views, (ii) the source relations, (iii) the available conversion, matching, and merging predicates.

2. We specify how to deal with tuples computed by the rewriting and possibly representing the same information. Typically we will have to combine tuples coming from different sources to obtain the tuples that contribute to Q.

The resulting query, which will be a disjunction of conjunctive queries, is the specification for the design of the mediator associated to Q. The above steps are discussed in more detail below.

4.4.1 Construction of the Rewriting

The computation of the rewriting is the most critical step of our method. Compared with other approaches, our query rewriting algorithm is complicated by the fact that we must consider both the constraints imposed by the conceptual level, and the Reconciliation Correspondences.

We describe now a rewriting algorithm suitable for our framework, that takes into account the above observation. First of all, we assume that the designer can specify an upper bound on the size of the conjunctive queries that can be used to compose the automatically generated query defining the mediator. Such an assumption is justified by considering that the size of the query directly affects the cost of computing the associated relation, and such a cost has to be limited due to several factors (e.g., the available time windows for loading and refreshment of a materialized view, in case the query is used to compute such a view).

The rewriting algorithm is based on generating *candidate rewritings* that are conjunctive queries limited in size by the bound specified by the designer, and verifying for each candidate rewriting

- whether the binding patterns for the correspondence predicates are respected;
- whether the rewriting is contained in the original query.

Each candidate rewriting that satisfies the conditions above contributes as a disjunct to the rewriting.

Checking whether the binding patterns are respected can be done in a standard way [50, 53]. On the other hand, verifying whether the rewriting is contained in the original query requires more attention, since we have to take into proper account the need of exploiting Reconciliation Correspondences in comparing different attribute values.

First of all, we have to pre-process the original adorned query as follows:

1. By introducing explicit equality atoms, we re-express the query (and the adornment) in such a way that all variables in the body and the head (excluding the adornment) are distinct. Then we replace each equality atom $X = Y$ between variables X and Y denoting attribute values with $\mathit{Equiv}([X],[Y])$. In this way we reflect the fact that we want to compare attribute values modulo representation in the relations. Notice that the case of an equality atom between two variables, one denoting an object and one denoting a value, can be excluded, since in this case the query would be inconsistent wrt the conceptual level.

2. We add to the query the atoms resulting from the adornment, considering each annotation as an atom: $X :: V$ is considered as the atom $V(X)$, and *Identify* is considered as a binary predicate that will be treated in a special way.

Also, when constructing the candidate rewriting, we have to take into account that the only way to compare attributes is through Reconciliation Correspondences. Therefore we require that in a candidate rewriting all variables are distinct, with the exception of those used in Reconciliation Correspondences, which may coincide with other variables. Notice that this is not a limitation, since the system provides by default the obvious Reconciliation Correspondences for equality.

We expand each atom in the candidate rewriting with the body of the query defining the predicate in the atom, including the adornment, as specified above. We have to take into account that atoms whose predicate is a correspondence predicate defined in terms of other correspondences, have to be expanded recursively, until all correspondence predicates have been substituted. We call the resulting query the *pre-expansion* of the candidate rewriting.

Then we add to the pre-expansion additional atoms derived by combining *Equiv* and *Identify* predicates as follows:

1. We add the symmetric and transitive closure of the *Equiv* atoms in the pre-expansion, i.e., we recursively add for each $\mathit{Equiv}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}])$ its symmetric $\mathit{Equiv}([\vec{\mathbf{y}}],[\vec{\mathbf{x}}])$, and for each pair $\mathit{Equiv}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}])$ and $\mathit{Equiv}([\vec{\mathbf{y}}],[\vec{\mathbf{z}}])$ the transitive composition $\mathit{Equiv}([\vec{\mathbf{x}}],[\vec{\mathbf{z}}])$. This reflects the fact that *Equiv* represents equality modulo representation in different relations.

2. We add $\mathit{Equiv}([X],[X])$ for each variable X denoting an attribute value. This is necessary to take into account that in the expanded query we have substituted equality atoms between variables denoting attribute values with *Equiv* atoms. Note that for tuples $\vec{\mathbf{x}}$ we do not need to consider atoms of the form $\mathit{Equiv}([\vec{\mathbf{x}}],[\vec{\mathbf{x}}])$.

3. We introduce equality atoms between variables denoting conceptual objects (not values) by adding $X = Y$ whenever we have either $\mathit{Identify}([\vec{\mathbf{x}}], X)$ and $\mathit{Identify}([\vec{\mathbf{x}}], Y)$, or $\mathit{Identify}([\vec{\mathbf{x}}], X)$, $\mathit{Identify}([\vec{\mathbf{y}}], Y)$ and $\mathit{Equiv}([\vec{\mathbf{x}}], [\vec{\mathbf{y}}])$. Indeed, *Equiv* reflects equality modulo representation, and *Identify* maps representations to the conceptual objects.

4. We propagate *Identify* through *Equiv*, by adding $\mathit{Identify}([\vec{\mathbf{x}}_1], X)$ whenever all variables in $\vec{\mathbf{x}}$ appear in the head and we have $\mathit{Equiv}([\vec{\mathbf{x}}_1], [\vec{\mathbf{x}}_2])$ and $\mathit{Identify}([\vec{\mathbf{x}}_2], X)$. This reflects the fact that $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{x}}_2$ are two different representations of the same conceptual object X.

We call the resulting query the *expansion* of the candidate rewriting.

Then, to decide whether the candidate rewriting is *correct* and hence can contribute to the final rewriting, we check if its expansion is contained in the pre-processed query, taking into account the conceptual level [11].

The rewriting of the original query is the union of all correct candidate rewritings. It can be shown that such a rewriting is *maximal* (wrt query containment) within the class of rewritings that are unions of conjunctive queries that respect the bound specified by the designer. This rewriting can be refined by using query containment to eliminate those correct candidate rewritings that are contained in others.

Observe that the query rewriting algorithm relies on the assumption that an upper bound for the length of the conjunctive query is defined: hence, it is possible that, by increasing the bound, one obtains a better rewriting.[2]

The computational complexity of the rewriting algorithm is dominated by the complexity of the containment test, which can be done in 2EXPTIME [11] wrt the size of the queries. However, one has to take into account that the size of queries can be neglected wrt the size of data at the sources (and in the materialized views, if present), and therefore the above bound does not represent a severe problem in practice.

Example 4.1 (cont.) We would like to obtain a specification in terms of a rewriting of the mediator that populates the relation $\mathsf{COMMONCHILD_{DW}}$. First, in the query defining $\mathsf{COMMONCHILD_{DW}}$, we eliminate common variables denoting attribute values by introducing *Equiv* atoms. We add the atoms resulting from the adornment, obtaining the pre-processed query:

$$
\begin{array}{rcl}
\mathsf{COMMONCHILD_{DW}}(S_f, S_m) & \leftarrow & \mathsf{Male}(F) \wedge \mathsf{ssn}(F, S1_f) \wedge \mathsf{CHILD}(F, C) \wedge \\
& & \mathsf{Female}(M) \wedge \mathsf{ssn}(M, S1_m) \wedge \mathsf{CHILD}(M, C) \wedge \\
& & \mathit{Equiv}([S_f], [S1_f]) \wedge \mathit{Equiv}([S_m], [S1_m]) \wedge \\
& & \mathsf{SSNString}(S_f) \wedge \mathsf{SSNString}(S_m) \wedge \\
& & \mathit{Identify}([S_f], F) \wedge \mathit{Identify}([S_m], M)
\end{array}
$$

Now, to obtain a rewriting of the above query we can exploit the queries associated to the relations FATHER_1 in source $\mathcal{S}_1$ and MOTHER_2 in source $\mathcal{S}_2$, taking into account that persons are represented differently in the two sources. Indeed, consider the following candidate rewriting:

$$
\begin{array}{rcl}
R(S_f, S_m) & \leftarrow & \mathsf{FATHER}_1(N_f, D_f, N_c, D_c),\ \mathsf{MOTHER}_2(S2_m, S_c), \\
& & \mathit{match}_{\mathit{person}}([N_c, D_c], S_c), \\
& & \mathit{convert}_{\mathit{person}}([N_f, D_f], S_f),\ \mathit{convert}_{\mathsf{SSNString}}([S2_m], [S_m])
\end{array}
$$

To check that R is a correct rewriting, we first substitute each atom in the body of the query

[2] Even without a bound on the length of the conjuncts in the rewriting, unions of conjunctive queries cannot exactly capture the original query in general. Indeed, to do so one would need to consider rewritings expressed in a query language that is at least NP-hard in data complexity [18].

by its definition, considering also the adornments, obtaining:

$$\begin{array}{rcl} R(S_f, S_m) & \leftarrow & \mathsf{Male}(F1) \wedge \mathsf{Person}(C1) \wedge \mathsf{CHILD}(F1, C1) \wedge \\ & & \mathsf{name}(F1, N_f) \wedge \mathsf{dob}(F1, D_f) \wedge \mathsf{name}(C1, N_c) \wedge \mathsf{dob}(C1, D_c) \wedge \\ & & \mathsf{NameString}(N_f) \wedge \mathsf{NameString}(N_c) \wedge \mathsf{Date}(D_f) \wedge \mathsf{Date}(D_c) \wedge \\ & & \mathit{Identify}([N_f, D_f], F1) \wedge \mathit{Identify}([N_c, D_c], C1) \wedge \\ & & \mathsf{Female}(M2) \wedge \mathsf{Person}(C2) \wedge \mathsf{CHILD}(M2, C2) \wedge \\ & & \mathsf{ssn}(M2, S2_m) \wedge \mathsf{ssn}(C2, S_c) \wedge \\ & & \mathsf{SSNString}(S2_m) \wedge \mathsf{SSNString}(S_c) \wedge \\ & & \mathit{Identify}([S2_m], M2) \wedge \mathit{Identify}([S_c], C2) \wedge \\ & & \mathit{Equiv}([N_c, D_c], [S_c]) \wedge \\ & & \mathsf{Person}(P3) \wedge \mathsf{name}(P3, N_c) \wedge \mathsf{dob}(P3, D_c) \wedge \mathsf{ssn}(P3, S_c) \wedge \\ & & \mathsf{NameString}(N_c) \wedge \mathsf{Date}(D_c) \wedge \mathsf{SSNString}(S_c) \wedge \\ & & \mathit{Identify}([N_c, D_c], P3) \wedge \mathit{Identify}([S_c], P3) \wedge \\ & & \mathit{Equiv}([N_f, D_f], [S_f]) \wedge \\ & & \mathsf{Person}(P4) \wedge \mathsf{name}(P4, N_f) \wedge \mathsf{dob}(P4, D_f) \wedge \mathsf{ssn}(P4, S_f) \wedge \\ & & \mathsf{NameString}(N_f) \wedge \mathsf{Date}(D_f) \wedge \mathsf{SSNString}(S_f) \wedge \\ & & \mathit{Identify}([N_f, D_f], P4) \wedge \mathit{Identify}([S_f], P4) \wedge \\ & & \mathit{Equiv}([S2_m], [S_m]) \wedge \mathsf{SSNString}(S2_m) \wedge \mathsf{SSNString}(S_m) \end{array}$$

Then we add to the body of the query the following atoms resulting from propagating *Identify* and *Equiv*

$$F_1 = P4 \wedge C1 = C2 \wedge C1 = P3 \wedge \mathit{Identify}([S_m], M2) \wedge \mathit{Identify}([S_f], F1)$$

plus an atom $\mathit{Equiv}([X], [X])$, for each variable X denoting an attribute value.

It is easy to check that the expanded candidate rewriting is indeed contained in the preprocessed query, which shows that the candidate rewriting is correct. ■

Example 4.2 (cont.) To obtain a specification of the mediator that populates the relation $\mathsf{SAMEINCOME_{DW}}$, we can pre-process the query defining such a relation obtaining:

$$\begin{array}{rcl} \mathsf{SAMEINCOME_{DW}}(S_1, S_2, I) & \leftarrow & \mathsf{Person}(P_1) \wedge \mathsf{ssn}(P_1, S_3) \wedge \mathsf{income}(P_1, I_1) \wedge \\ & & \mathsf{Person}(P_2) \wedge \mathsf{ssn}(P_2, S_4) \wedge \mathsf{income}(P_2, I_2) \wedge \\ & & \mathit{Equiv}([S_1], [S_3]) \wedge \mathit{Equiv}([S_2], [S_4]) \wedge \mathit{Equiv}([I_1], [I_2]) \wedge \\ & & \mathsf{SSNString}(S_1) \wedge \mathsf{SSNString}(S_2) \wedge \mathsf{IncomePerYear}(I) \wedge \\ & & \mathit{Identify}([S_1], P_1) \wedge \mathit{Identify}([S_2], P_2) \end{array}$$

A candidate rewriting in terms of the queries defining INCOME_1 in source $\mathcal{S}_1$ and INCOME_2 in source $\mathcal{S}_2$ is:

$$\begin{array}{rcl} R_1(S_1, S_2, I) & \leftarrow & \mathsf{INCOME}_1(S_m, I_m) \wedge \mathsf{INCOME}_2(S_y, I_y) \wedge \\ & & match_{income}([I_m], [I_y]) \wedge \\ & & convert_{\mathsf{IncomePerYear}}([I_y], [I]) \wedge \\ & & convert_{\mathsf{SSNString}}([S_m], [S_1]) \wedge convert_{\mathsf{SSNString}}([S_y], [S_2]) \end{array}$$

where $match_{income}$ is the Matching Correspondence automatically generated from $convert_{income}$, and the other Conversion Correspondences are automatically generated for the proper domains. We can again check that the expansion of the rewriting is contained in the pre-processed query, by taking into account that the schema, which the queries refers to, implies that Male and Female are sub-entities of Person.

The above correct candidate rewriting corresponds to perform the join between relations INCOME_1 and INCOME_2. Considering also the other possible joins between the two relations we obtain the following rewriting R:

$$
\begin{array}{rcl}
R_1(S_1, S_2, I) & \leftarrow & \mathsf{INCOME}_1(S_m, I_m) \wedge \mathsf{INCOME}_2(S_y, I_y) \wedge \\
& & match_{income}([I_m], [I_y]) \wedge \\
& & convert_{\mathsf{IncomePerYear}}([I_y], [I]) \wedge \\
& & convert_{\mathsf{SSNString}}([S_m], [S_1]) \wedge convert_{\mathsf{SSNString}}([S_y], [S_2]) \\
& & \vee \\
& & \mathsf{INCOME}_2(S_y, I_y) \wedge \mathsf{INCOME}_1(S_m, I_m) \wedge \\
& & match_{income}([I_m], [I_y]) \wedge \\
& & convert_{\mathsf{IncomePerYear}}([I_y], [I]) \wedge \\
& & convert_{\mathsf{SSNString}}([S_m], [S_1]) \wedge convert_{\mathsf{SSNString}}([S_y], [S_2]) \\
& & \vee \\
& & \mathsf{INCOME}_1(S_m, I_m) \wedge \mathsf{INCOME}_1(S1_m, I1_m) \wedge \\
& & match_{\mathsf{IncomePerMonth}}([I_m], [I1_m]) \wedge \\
& & convert_{income}([I_m], [I]) \wedge \\
& & convert_{\mathsf{SSNString}}([S_m], [S_1]) \wedge convert_{\mathsf{SSNString}}([S1_m], [S_2]) \\
& & \vee \\
& & \mathsf{INCOME}_2(S_y, I_y) \wedge \mathsf{INCOME}_2(S1_y, I1_y) \wedge \\
& & match_{\mathsf{IncomePerYear}}([I_y], [I1_y]) \wedge \\
& & convert_{\mathsf{IncomePerYear}}([I_y], [I]) \wedge \\
& & convert_{\mathsf{SSNString}}([S_y], [S_1]) \wedge convert_{\mathsf{SSNString}}([S1_y], [S_2])
\end{array}
$$

■

Observe that it may happen that no rewriting for the query exists. One reason for this may be that the available relations do not contain enough information to populate the relation defined by the query. In this case a further analysis of the source is required. It may also be the case that the relations do in fact contain the information needed, but the available reconciliation programs are not able to convert the data in a representation suitable to answer the query. Formally, this is reflected in the fact that appropriate Reconciliation Correspondences are missing. In this case our rewriting algorithm can be used by the designer to acquire indications on which are the required Reconciliation Correspondences, and hence the associated programs that must be added to generate the mediator.

4.4.2 Combining Tuples Coming from Different Sources

Since the rewriting constructed as specified above is in general a disjunction, we must address the problem of combining the results of several queries. A similar problem arises in other approaches [28], where the result of approximate joins may require a specification of a construction operation. In order to properly define the result of the query, we introduce the notion of *combine-clause*. In particular, if the query r computed by the rewriting is constituted by more than one disjunct, then the algorithm associates to r a suitable set of so-called merging clauses, taking into account that the answers to the different disjuncts of the query may contain tuples that represent the same real world entity or the same value. A *combine-clause* is an expression of the form

$$
\begin{array}{l}
\texttt{combine } \textit{tuple-spec}_1 \texttt{ and } \cdots \texttt{ and } \textit{tuple-spec}_n \\
\texttt{such that } \textit{combination-condition} \\
\texttt{into } \textit{tuple-spec}_{t_1} \texttt{ and } \cdots \texttt{ and } \textit{tuple-spec}_{t_m}
\end{array}
$$

where *tuple-spec*$_i$ denotes a tuple returned by the i-th disjunct of r, *combination-condition* specifies how to combine the various tuples denoted by *tuple-spec*$_1$,...,*tuple-spec*$_n$, and *tuple-spec*$_{t_1}$,...,*tuple-spec*$_{t_m}$ denote the tuples resulting from the combination that are inserted in the relation defined by the query.

We observe that the rewriting algorithm is able to generate one combine-clause template for each pair of disjuncts that are not disjoint. Starting from such templates, the designer may either specify the `such that` and the `into` parts, depending on the intended semantics, or change the templates in order to specify a different combination plan (for example for combining three disjuncts, rather than three pairs of disjuncts).

4.4.3 Refining the Rewriting

As already mentioned, the rewriting returned by the algorithm can be refined by eliminating certain conjuncts from the union of conjunctive queries according to suitable criteria for populating relations. In particular, such criteria may be determined by factors that affect the quality of the data in the source relations and in the materialized views, such as completeness, accuracy, confidence, freshness, etc. In practice, a convenient way to characterize such quality factors is by providing ad hoc information in a meta-repository [7, 36, 37].

While the goal of the present work is to provide a language to represent transformations, thus not requiring an explicit introduction of *meta-level descriptions*, there are certain aspects of the data integration process that can take advantage of a meta-information approach. More specifically, together with the actual data stored in the source, the wrapper can provide meta-data, both at relation-level and at tuple-level, which can be suitably exploited in refining the rewriting. As an example, consider the case where a materialized view is available. The materialized view can be described as any other source, being always preferred with respect to other sources because of its higher accuracy and confidence.

5 Reconciliation in GAV

When the mapping is specified following the global-as-view approach, the global schema is defined in terms of the data sources. In our framework, we associate a query over the sources to each concept of the global schema, with the intended meaning of specifying how to retrieve the data corresponding to such concept starting from the data at the sources. More precisely, given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, the mapping $\mathcal{M}$ associates queries to the elements of $\mathcal{G}$ as follows:

- The mapping associates a query of arity 1 to each entity of $\mathcal{G}$.
- The mapping associates a query of arity 2 to each attribute A defined for an entity in $\mathcal{G}$. Intuitively, if the query retrieves (c, d) from the sources, this means that d is a value of the attribute A of the entity instance c.
- The mapping associates a query of arity n to each relationship R of arity n in $\mathcal{G}$. Intuitively, if the query retrieves the tuple $(c_1, \ldots, c_n)$ from the sources, this means that $(c_1, \ldots, c_n)$ is an instance of R.
- The mapping associates a query of arity $n+1$ to each attribute A defined for a relationship R of arity n in $\mathcal{G}$. Intuitively, if the query retrieves $(c_1, \ldots, c_n, d)$ from the sources, this means that d is a value of the attribute A of the relationship instance $(c_1, \ldots, c_n)$.

Notice that, as in the $\mathcal{DLR}$ approach to data integration described in Section 4, the views that define the mapping relate the conceptual representation of the global schema with the logical representation of the sources. Thus, to specify how conceptual objects in the queries are

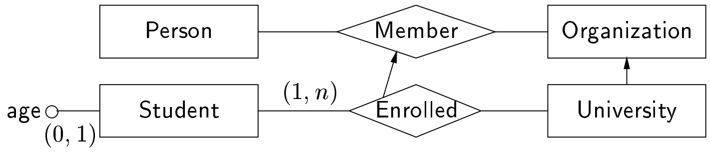

Figure 2: Global schema of Example 5.1

represented by tuples of logical values, it is necessary, also in this case, to specify additional information in the mapping assertions. We argue that this is possible by adapting to the GAV mapping specification the mechanism introduced to cope with this problem in the $\mathcal{DLR}$ approach to data integration, i.e., by adding a suitable *adornment* clause to each query. Nonetheless, to keep things simple, we do not make use of adornment clauses in the GAV mapping assertions, and actually treat entity, relationship, and attribute symbols of the conceptual global schema as relation symbols of a relational model.

Furthermore, in the following we consider the particular case in which queries over the conceptual model are *conjunctive queries* (CQ), rather than union of conjunctive queries. Intuitively, a conjunctive query is a UCQ where the body consists of a single disjunct.

Example 5.1 Figure 2 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, 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 $\mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3, \mathsf{S}_4, \mathsf{S}_5, \mathsf{S}_6, \mathsf{S}_7, \mathsf{S}_8$, and that the mapping $\mathcal{M}_1$ is as follows:

$$
\begin{array}{rcl}
\mathsf{Person}(X) & \leftarrow & \mathsf{S}_1(X) \\
\mathsf{Organization}(X) & \leftarrow & \mathsf{S}_2(X) \\
\mathsf{Member}(X,Y) & \leftarrow & \mathsf{S}_7(X,Z), \mathsf{S}_8(Z,Y)
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{Student}(X) & \leftarrow & \mathsf{S}_3(X,Y) \vee \mathsf{S}_4(X,Z) \\
\mathsf{University}(X) & \leftarrow & \mathsf{S}_5(X) \\
\mathsf{Enrolled}(X,Y) & \leftarrow & \mathsf{S}_4(X,Y) \\
\mathsf{age}(X,Y) & \leftarrow & \mathsf{S}_3(X,Y) \vee \mathsf{S}_6(X,Y,Z)
\end{array}
$$

■

From the definition of the semantics of a data integration system, given in Section 3, it is easy to see that, in our framework, given a source database $\mathcal{D}$, different situations are possible:

1. Several legal global databases exist. This happens, for example, when the data at the sources retrieved by the queries associated to the global concepts do not satisfy the is-a relationships or the mandatory participation constraints 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 ways yields a legal global database.

Example 5.2 Referring to Example 5.1, consider a source database $\mathcal{D}_2$, where S_1 stores p_1 and p_2, S_2 stores o_1, S_5 stores u_1, and S_3 stores t_1, and the pairs (p_1, o_1) and (p_2, u_1) are in the join between S_7 and S_8. By the mapping $\mathcal{M}_1$, it follows that in every legal database of $\mathcal{I}_1$, $p_1, p_2 \in \mathsf{Person}$, $(p_1, o_1), (p_2, u_1) \in \mathsf{Member}$, $o_1 \in \mathsf{Organization}$, $t_1 \in \mathsf{Student}$, $u_1 \in \mathsf{University}$. Moreover, since $\mathcal{G}_1$ specifies that Student has a mandatory participation in the relationship Enrolled, in every legal database for $\mathcal{I}_1$, t_1 *must* be enrolled in a certain university. The key point is that nothing is said in $\mathcal{D}_2$ about *which* university, and therefore we have to accept as legal all databases for $\mathcal{I}_1$ that differ in the university in which t_1 is enrolled. ■

2. No legal global database exists. This happens, in particular, when the data at the sources retrieved by the queries associated to the elements of the global schema do not satisfy the functional attribute constraints.

Example 5.3 Referring again to Example 5.1, consider a source database $\mathcal{D}_1$, where S_3 stores the tuple (t_1, a_1), and S_6 stores the tuple (t_1, a_2, v_1). The query associated to age by the mapping $\mathcal{M}_1$ specifies that, in every legal database of $\mathcal{I}_1$ both tuples should belong to the extension of age. However, age is a functional attribute in $\mathcal{G}_1$, and therefore no legal database exists for the data integration system $\mathcal{I}_1$. ■

The first problem shows that the issue of query answering with incomplete information arises even in the global-as-view approach to data integration. Indeed, the existence of multiple global databases for the data integration system implies that query processing cannot simply reduce to evaluating the query over a single database. Rather, we should in principles take *all* possible legal global databases into account when answering a query.

It is interesting to observe that there are at least two different strategies to simplify the setting, and overcome this problem:

1. Data integration systems usually adopt a simpler data model (often, a plain relational data model) for expressing the global schema. In this case, the data retrieved from the sources trivially fits into the schema, and can be directly considered as the unique database to be processed during query answering.

2. The queries associated to the elements of the global schema are often considered as exact. In this case, analogously to the previous one, it is easy to see that the only global database to be considered is the one formed by the data retrieved by the source. However, when data at the sources do not obey all semantic conditions that are implicit in the conceptual global schema, this single database is not coherent with the global schema, and the data integration system is inconsistent. This implies that query answering is meaningless. We argue that, in the usual case of autonomous, heterogeneous sources, it is very unlikely that data fit in the global schema, and therefore, this approach is too restrictive, in the sense that the data integration system would be often inconsistent.

The fact that the problem of incomplete information is overlooked in current approaches can be explained by observing that traditional data integration systems follow one of the above mentioned simplifying strategies: they either express the global schema as a set of plain relations, or consider the sources as exact (see, for instance, [20, 46, 9]). Hence, it is commonly assumed that, in order to answer a query posed over the global schema, it is sufficient to evaluate the query over the retrieved database, i.e. the database obtained by populating each global concept according to the corresponding view over the sources. More formally, given a source database $\mathcal{D}$, the *retrieved global database* $ret(\mathcal{I}, \mathcal{D})$ is obtained by evaluating, for each concept C of the global schema, the associated view V_C over $\mathcal{D}$. Evaluating the query over $ret(\mathcal{I}, \mathcal{D})$ is equivalent to *unfold* each atom of the original query with the corresponding definition. On the contrary, our framework for data integration considers the more general setting where the global schema is expressed in terms of a conceptual model, and sources are considered sound (but not necessarily complete). Example 5.2 shows that, in this case, we have to account for multiple global databases, and the results described in the next subsection demonstrate that unfolding is in general not sufficient to process a query, and show how to cope with this issue in this setting.

With regard to the second problem, we have shown that, when an attribute A of an entity (or relationship) E is functional, and such constraint is violated in the global schema by the data retrieved from the sources, we have that, according to the definition of semantics of a data integration system, no legal global database exists. Intuitively, this happens because the assumption of sound views (but this holds also for exact views) does not allow us to disregard tuples with different values for A and the same instance for E. A more general approach would be to provide a formalization that is able to provide the set of certain answers to a query over the global schema even when the data at the sources are mutually inconsistent. The basic idea is to consider those global databases that satisfy the integrity constraints in the global schema and that approximate the satisfaction of the mapping $\mathcal{M}$, i.e., that are *as much sound as possible*. One way to formalize this idea is to distinguish between *strictly-sound* mappings, as the ones considered before, and *loosely-sound* mappings, in which the assumption of soundness is suitably relaxed.

In the following we first deal only with the the problem of source incompleteness, and provide a procedure to suitably answer a user's query in our framework, in which is-a and mandatory participation constraints are expressed on the global schema. Data reconciliation in this case is performed under the first-order semantics introduced in section 3, henceforth called *strictly-sound semantics*. Then we also consider the problem of mutual inconsistency of the data stored at different sources, and define a new semantics, called the *loosely-sound semantics*, that allows us to reconcile data in this situation. Moreover, we provide a technique to answer user's queries in our framework when the functional attribute constraints defined in the global schema are violated by the data retrieved from the sources.

5.1 Data Reconciliation under the strictly-sound semantics

In this section we present an algorithm for computing the set of certain answers to queries posed to a data integration system in the presence of is-a and mandatory participation constraints. The key feature of the algorithm is to reason about both the query and the conceptual global schema in order to infer which tuples satisfy the query in all legal databases of the data integration system. Thus, the algorithm does not simply unfold the query on the basis of the mapping, as usually done in data integration systems based on the global-as-view approach, but takes into account the several databases that are legal for the data integration system according to the strictly-sound semantics. Indeed, we now show that a simple unfolding strategy does not work in our setting.

Example 5.4 Referring to Example 5.2, consider the query Q_1 to $\mathcal{I}_1$:

$$Q_1(x) \leftarrow \mathsf{Member}(x, y), \mathsf{University}(y),$$

and suppose we simply unfold the query Q_1 in the standard way, by substituting each atom with the query that $\mathcal{M}_1$ associates to the element in the atom. Then we get the query

$$Q(x) \leftarrow \mathsf{S}_7(x, z), \mathsf{S}_8(z, y), \mathsf{S}_5(y)$$

If we evaluate this query over $\mathcal{D}_2$, we get $\{p_2\}$ as result. On the other hand, it is easy to see that the real set of certain answers to Q_1 with respect to $\mathcal{I}_1$ and $\mathcal{D}_2$ is $\{p_2, t_1\}$, and by processing the query via unfolding we miss the certain answer t_1. Actually, although $\mathcal{D}_2$ does not indicate which university t_1 is enrolled in, the semantics of $\mathcal{I}_1$ specifies that t_1 is enrolled in *a* university in all legal database for $\mathcal{I}_1$. Since Member is a generalization of Enrolled, this implies that t_1 is in the set of certain answers to Q_1 with respect to $\mathcal{I}_1$ and $\mathcal{D}_2$. ■

Next we illustrate our algorithm for computing all certain answers. The algorithm is able to add more answers to those directly extracted from the sources, by exploiting the semantic conditions expressed in the conceptual global schema.

Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be an integration system, let $\mathcal{D}$ be a source database, and let Q be a query over the global schema $\mathcal{G}$. The algorithm is constituted by three major steps.

1. From the query Q, obtain a new query $exp_{\mathcal{G}}(Q)$ over the elements of the global schema $\mathcal{G}$ in which the knowledge in $\mathcal{G}$ that is relevant for Q has been compiled in.

2. From $exp_{\mathcal{G}}(Q)$, compute the query $unf_{\mathcal{M}}(exp_{\mathcal{G}}(Q))$, by unfolding $exp_{\mathcal{G}}(Q)$ on the basis of the mapping $\mathcal{M}$. The unfolding simply substitutes each atom of $exp_{\mathcal{G}}(Q)$ with the query associated by $\mathcal{M}$ to the element in the atom. The resulting $unf_{\mathcal{M}}(exp_{\mathcal{G}}(Q))$ is a query over the source relations.

3. Evaluate the query $unf_{\mathcal{M}}(exp_{\mathcal{G}}(Q))$ over the source database $\mathcal{D}$.

The last two steps are quite obvious. Instead, the first one requires to find a way to compile into the query the semantic relations holding among the concepts of the global schema $\mathcal{G}$. Such semantic relations can indeed be crucial for inferring the complete set of certain answers.

The basic idea to do so is that the relations among the elements in $\mathcal{G}$ can be captured by a suitable *rule base* $\mathcal{R}_{\mathcal{G}}$. To build $\mathcal{R}_{\mathcal{G}}$, we introduce a new predicate P' (called primed predicate) for each predicate P associated to an element P of $\mathcal{G}$. Then, from the semantics of the ER schema we devise the following rules (expressed in Logic Programming notation [47]):

- for each entity E, attribute A and relationship R in $\mathcal{G}$, we have:

$$\begin{array}{l} E'(X) \leftarrow E(X) \\ A'(X,Y) \leftarrow A(X,Y) \\ R'(X_1,\ldots,X_n) \leftarrow R(X_1,\ldots,X_n) \end{array}$$

- for each is-a relation between E and E_i, or between R and R_i, we have:

$$\begin{array}{l} E'_i(X) \leftarrow E'(X) \\ R'_i(X_1,\ldots,X_n) \leftarrow R'(X_1,\ldots,X_n) \end{array}$$

- for each attribute A for an entity E or a relationship R, we have:

$$\begin{array}{l} E'(X) \leftarrow A'(X,Y) \\ R'(X_1,\ldots,X_n) \leftarrow A'(X_1,\ldots,X_n,Y) \end{array}$$

- for each relationship R involving an entity E_i as i-th component, we have:

$$E'_i(X_i) \leftarrow R'(X_1,\ldots,X_i,\ldots,X_n)$$

- for each mandatory participation of an entity E in a relationship R_j via the ER-role i, we have:

$$R'_j(f_1(X),\ldots,f_{i-1}(X),X,f_{i+1}(X),\ldots,f_n(X)) \leftarrow E'(X)$$

  where f_i are fresh Skolem functions [47].

- for each mandatory attribute A for an entity E or a relationship R in an attribute definition of $\mathcal{G}$, we have:

$$\begin{array}{l} A'(X,f(X)) \leftarrow E'(X) \\ A'(X_1,\ldots,X_n,f(X)) \leftarrow R'(X_1,\ldots,X_n) \end{array}$$

  where f is a fresh Skolem function.

Once we have defined such a rule base $\mathcal{R}_{\mathcal{G}}$, we can use it to generate the query $exp_{\mathcal{G}}(Q)$ associated to the original query Q. This is done as follows:

1. First, we rewrite Q by substituting each predicate P in the body $body(Q)$ of Q with P'. We denote by Q' the resulting query. In the following we call "primed atom" every atom whose predicate is primed.

2. Then we build a *partial resolution tree* for Q', i.e., a tree having each node labeled by a conjunctive query q, with one of the atoms in $body(q)$ marked as "*selected*", obtained as follows.

   (a) The root is labeled by Q', and has marked as selected any (primed) atom in $body(Q')$ (for example the first in left-to-right order).

   (b) Except if condition (2c) below is satisfied, a node, labeled by a query q having a "selected" atom α, has one child for each rule r in $\mathcal{R}_{\mathcal{G}}$ such that there exists a most general unifier[3] $mgu(\alpha, head(r))$ between the atom α and the head $head(r)$ of the rule r, such that the variables appearing in the head of Q' are to assigned to terms involving Skolem functions. Each of such children has the following properties:
      - it is labeled by the query obtained from q by replacing the atom α with $body(r)$ and by substituting the variables with $mgu(\alpha, head(r))$;
      - it has as marked "selected" one of the primed atoms (for example the first in left-to-right order).

   (c) If a node d that is labeled by a query q and there exists a predecessor d' of d labeled by a query q' and a substitution θ of the variables of q' that makes q' equal to q, then d has a single child, which is labeled by the empty query (a query whose body is false).

3. Finally we return as result the query $exp_{\mathcal{G}}(Q)$ formed as the union of all non-empty queries in the leaves of the partial resolution tree and that do not contain Skolem functions.

The following three observations are crucial for characterizing both the termination and the correctness of our algorithm:

- The termination of the construction of the tree, and thus of the entire algorithm, is guaranteed by the condition (2c) and by the observation that all the rules in $\mathcal{R}_{\mathcal{G}}$ have a single atom in the body.

- By exploiting results on partial evaluation of logic programs (see [24]), it can be shown that $exp_{\mathcal{G}}(Q)$ is equivalent to the original query Q with respect to the global schema $\mathcal{G}$, that is, for each database $\mathcal{B}$ that is legal for $\mathcal{G}$, the evaluation of Q yields the same result as $exp_{\mathcal{G}}(Q)$, i.e., $Q^{\mathcal{B}} = (exp_{\mathcal{G}}(Q))^{\mathcal{B}}$.

- The query $exp_{\mathcal{G}}(Q)$ returned by the algorithm is a union of conjunctive queries. Each disjunct of $exp_{\mathcal{G}}(Q)$ is a conjunctive query over the predicates of the global schema, i.e., the elements that have an associated query over the sources by virtue of the mapping.

The above observations imply that, if we evaluate $unf_{\mathcal{M}}(exp_{\mathcal{G}}(Q))$ over the source database $\mathcal{D}$, we get exactly the set of certain answers $Q^{\mathcal{I},\mathcal{D}}$ of Q with respect to $\mathcal{I}$ and $\mathcal{D}$.

With regard to the characterization of the computational complexity of the algorithm, we observe that the number of disjuncts in $exp_{\mathcal{G}}(Q)$ can be exponential in the number of rules in the rule base $\mathcal{R}_{\mathcal{G}}$ (and therefore in the size of the global schema $\mathcal{G}$), and in the number of variables in the original query Q. Note, however, that this bound is independent of the size of $\mathcal{D}$, i.e., the size of data at the sources. We remind the reader that the evaluation of a union of conjunctive queries can be done in time polynomial with respect to the size of the data. Since $exp_{\mathcal{G}}(Q)$ is a union of conjunctive queries, we can conclude that, if the queries associated by $\mathcal{M}$ to the elements of $\mathcal{G}$ can be evaluated in polynomial time in the size of the data at the sources, then evaluating $unf_{\mathcal{M}}(exp_{\mathcal{G}}(Q))$ over $\mathcal{D}$ is also polynomial in the size of the data at the sources. It follows that our query answering algorithm is polynomial with respect to data complexity.

[3] We recall that given two atoms α and β the most general unifier $mgu(\alpha, \beta)$ is a most general substitution for the variables in α and β that makes α and β equal [47].

Example 5.5 Referring again to Example 5.4, it is possible to see that, by evaluating the unfolding of the query returned by the algorithm, the whole set of certain answers to Q_1 with respect to $\mathcal{I}_1$ and $\mathcal{D}_2$ is obtained. In particular, t_1 is obtained by processing the rule $\mathsf{Member}'(X,Y) \leftarrow \mathsf{Enrolled}'(X,Y)$, which takes into account that Member is a generalization of Enrolled and the rule $\mathsf{Enrolled}'(X,f(X)) \leftarrow \mathsf{Student}'(X)$, which expresses the mandatory participation of Student in Enrolled.

5.2 Data Reconciliation under the loosely-sound semantics

Consider now the situation in which there exists no global database that both is coherent with the global schema and satisfies the mapping wrt a given source database: in our setting, this corresponds to a case in which the retrieved database violates the functional attribute constraints in the global schema. Under the strictly-sound semantics, this means that given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and a source database $\mathcal{D}$, there are no legal databases for $\mathcal{I}$ wrt $\mathcal{D}$.

As we already said, in order to provide the set of certain answers to a query also in this situation, we introduce a new notion of mapping satisfaction in which the sound view assumption is suitably relaxed. To simplify the discussion, in the following we indicate the mapping as a set of assertions of the form $\langle C, V_C \rangle$, where C is a concept of the global schema and V_C is the associated view that provides its definition in terms of the sources. Then, given a source database $\mathcal{D}$ for $\mathcal{I}$ and a mapping $\mathcal{M} = \{\langle C_1, V_1 \rangle \ \ldots \ \langle C_n, V_n \rangle\}$, we define an ordering between the global databases for $\mathcal{I}$.

If $\mathcal{B}_1$ and $\mathcal{B}_2$ are two databases coherent with $\mathcal{G}$, we say that $\mathcal{B}_1$ is *better* than $\mathcal{B}_2$ wrt $\mathcal{D}$, denoted as $\mathcal{B}_1 \gg_{\mathcal{D}} \mathcal{B}_2$, iff one of the two following conditions holds:

1. there exists $i \in \{1, \ldots, n\}$ such that
   - $(C_i^{\mathcal{B}_1} \cap V_i^{\mathcal{D}}) \supset (C_i^{\mathcal{B}_2} \cap V_i^{\mathcal{D}})$, and
   - $(C_j^{\mathcal{B}_1} \cap V_j^{\mathcal{D}}) \supseteq (C_j^{\mathcal{B}_2} \cap V_j^{\mathcal{D}})$ for $j = 1, \ldots, n$;
2. $(C_k^{\mathcal{B}_1} \cap V_k^{\mathcal{D}}) = (C_k^{\mathcal{B}_2} \cap V_k^{\mathcal{D}})$ for $k = 1, \ldots, n$ and there exists $i \in \{1, \ldots, n\}$ such that
   - $(C_i^{\mathcal{B}_1} - V_i^{\mathcal{D}}) \subset (C_i^{\mathcal{B}_2} - V_i^{\mathcal{D}})$, and
   - $(C_j^{\mathcal{B}_1} - V_j^{\mathcal{D}}) \subseteq (C_j^{\mathcal{B}_2} - V_j^{\mathcal{D}})$ for $j = 1, \ldots, n$.

Intuitively, this means that the retrieved portion of the global database, i.e., the subset that is computed by the views, is (1) greater in $\mathcal{B}_1$ than in $\mathcal{B}_2$, i.e., $\mathcal{B}_1$ approximates the sound mapping better than $\mathcal{B}_2$, or (2) is the same in $\mathcal{B}_1$ and in $\mathcal{B}_2$, but the tuples that are not part of the retrieved subset are less in $\mathcal{B}_1$ than in $\mathcal{B}_2$. In other words, $\mathcal{B}_1 \gg_{\mathcal{D}} \mathcal{B}_2$ iff $\mathcal{B}_1$ is "closer" to the retrieved database than $\mathcal{B}_2$. It is easy to verify that the relation $\gg_{\mathcal{D}}$ is a partial order. With this notion in place, we say that a global database $\mathcal{B}$ coherent with $\mathcal{G}$ satisfies the mapping, considered now loosely-sound, if $\mathcal{B}$ is *maximal* wrt $\gg_{\mathcal{D}}$, i.e., for no other global database $\mathcal{B}'$ coherent with $\mathcal{G}$, we have that $\mathcal{B}' \gg_{\mathcal{D}} \mathcal{B}$. Hence, with reference to the definition given in Section 3, a global database for $\mathcal{I}$ is said to be *legal* wrt $\mathcal{D}$ under a loosely-sound semantics, if $\mathcal{B}$ is coherent with $\mathcal{G}$ and is maximal wrt $\gg_{\mathcal{D}}$.

It is immediate to verify that, if there exists a legal database for $\mathcal{I}$ wrt $\mathcal{D}$ under the strictly-sound semantics, then the strictly-sound and the loosely-sound semantics coincide, in the sense that, for each query Q, the set $Q^{\mathcal{I},\mathcal{D}}$ of certain answers computed under the strictly-sound semantics coincides with the set of certain answers computed under the loosely-sound semantics.

On the other hand, in the cases in which no legal database exists under the strictly-sound semantics, it is easy to see that there always exists a legal database for $\mathcal{I}$ wrt $\mathcal{D}$ under a loosely-sound semantics, because we are allowed to eliminate tuples from the retrieved database $ret(\mathcal{I}, \mathcal{D})$ in order to satisfy the constraints, and functional attribute, is-a, and mandatory participation constraints can always be satisfied by suitably restricting the set of tuples in the database. However, the semantics implies that the legal databases are the ones that are "as close as possible" to $ret(\mathcal{I},\mathcal{D})$, thus we have to consider only databases coherent with the constraints and that "minimize" elimination of tuples from the $ret(\mathcal{I},\mathcal{D})$. Since is-a and mandatory participation constraints can be satisfied by adding new tuples in the database, elimination of tuples is only forced by functional attribute constraints satisfaction. In fact, it can be shown that it is possible to compute the certain answers to a query Q through the following two-step process:

1. first, identify the databases corresponding to the legal databases for $\mathcal{I}'$ wrt $\mathcal{D}$, where $\mathcal{I}'$ is obtained from $\mathcal{I}$ by eliminating all is-a and mandatory participation constraints in $\mathcal{G}$. It is immediate to see that each such database $\mathcal{B}'$ is "contained" in $ret(\mathcal{I},\mathcal{D})$, i.e. if $t \in C^{\mathcal{B}'}$ then $t \in C^{ret(\mathcal{I},\mathcal{D})}$ for each t and for each C in $\mathcal{G}$;

2. t is a certain answer of Q wrt $\mathcal{I}$ and $\mathcal{D}$ iff $t \in Q^{\mathcal{B}'}$ for each database $\mathcal{B}'$ computed in the previous step.

As for the first of the above steps, we resort to DATALOG$^\neg$ under stable model semantics [41, 27], a well-known extension of DATALOG that allows for using negation in the body of program rules. In particular, we define a DATALOG$^\neg$ program $P(\mathcal{I},\mathcal{D})$ that allows for computing the legal databases for $\mathcal{I}'$ wrt $\mathcal{D}$. The DATALOG$^\neg$ program $P(\mathcal{I},\mathcal{D})$ is obtained by adding to the set of facts $\mathcal{D}$ the following set of rules:

- for each assertion $\langle C, V_C\rangle \in \mathcal{M}$, with

$$V_C = C(\vec{\mathbf{x}}) \ \leftarrow \ conj_1(\vec{\mathbf{x}},\vec{\mathbf{y}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{x}},\vec{\mathbf{y}}_m)$$

  the rules

$$\begin{array}{rcl} C_{\mathcal{D}}(\vec{\mathbf{x}}) & \leftarrow & conj_1(\vec{\mathbf{x}},\vec{\mathbf{y}}_1) \\ & \cdots & \\ C_{\mathcal{D}}(\vec{\mathbf{x}}) & \leftarrow & conj_m(\vec{\mathbf{x}},\vec{\mathbf{y}}_m) \end{array}$$

- for each functional attribute $A \in \mathcal{G}$, the rules

$$\begin{array}{rcl} A(\vec{\mathbf{x}},Y) & \leftarrow & A_{\mathcal{D}}(\vec{\mathbf{x}},Y)\ ,\ not\ \overline{A}(\vec{\mathbf{x}},Y) \\ \overline{A}(\vec{\mathbf{x}},Y) & \leftarrow & A(\vec{\mathbf{x}},Z)\ ,\ Y \neq Z \end{array}$$

  where $\vec{\mathbf{x}}$ in $a(\vec{\mathbf{x}},Y)$ can be either a single variable X when A is an attribute of an entity E, or an n-tuple $X_1,\ldots,X_n$ when A is an attribute of a relationship R of arity n. In the first case X corresponds to the entity E, whereas in the second case $X_1,\ldots,X_n$ correspond to the entities that participate to R.

Informally, for each concept C in $\mathcal{G}$, $P(\mathcal{I},\mathcal{D})$ contains a concept $C_{\mathcal{D}}$ that represents $C^{ret(\mathcal{I},\mathcal{D})}$, and for each functional attribute A, it contains (i) a further attribute A that represents a subset of $A^{ret(\mathcal{I},\mathcal{D})}$ that is consistent with the fact that A is functional, and (ii) an auxiliary attribute $\overline{A}$. The above rules force each stable model M of $\mathcal{P}(\mathcal{I},\mathcal{D})$ to be such that A^M is a maximal subset of tuples from $A^{ret(\mathcal{I},\mathcal{D})}$ that are consistent with the constraint that makes A to be functional.

It can be shown that each stable model M for $P(\mathcal{I},\mathcal{D})$ corresponds to a legal database $\mathcal{B}'$ for $\mathcal{I}'$ wrt $\mathcal{D}$, in the sense that, for each $C \in \mathcal{G}$, $C^{\mathcal{B}'} = \{t \mid t \in C^M\}$, and conversely, for each legal database $\mathcal{B}'$ for $\mathcal{I}'$ wrt $\mathcal{D}$ there exists a stable model M for $P(\mathcal{I},\mathcal{D})$ such that, for each $C \in \mathcal{G}$, $\mathcal{B}' = \{t \mid t \in C^M\}$.

As for the second of the above steps, we make use of the query reformulation algorithm presented in the above subsection that transforms the query q into a query $exp_{\mathcal{G}}(q)$. By adding this query to the program $P(\mathcal{I},\mathcal{D})$, we obtain a DATALOG$^\neg$ program that allows us to compute the certain answers to the original query.

Theorem 5.6 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, let Q be a query posed to $\mathcal{I}$, $\mathcal{D}$ be a source database for $\mathcal{I}$, and t be a tuple of constants of the same arity as Q. Then, $t \in Q^{\mathcal{I},\mathcal{D}}$ if and only if $t \in q^M$ for each stable model M of the* DATALOG$^\neg$ *program $P(\mathcal{I}, \mathcal{D}) \cup \{exp_{\mathcal{G}}(Q)\}$.*

Finally, we are able to characterize the computational complexity of the problem of computing certain answers to queries in our data integration setting.

Theorem 5.7 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, let Q be a query posed to $\mathcal{I}$, $\mathcal{D}$ be a source database for $\mathcal{I}$, and t be a tuple of constants of the same arity as Q. The problem of deciding whether $t \in Q^{\mathcal{I},\mathcal{D}}$ is coNP-complete wrt data complexity.*

Proof sketch. Membership in coNP follows from Theorem 5.6, and from the fact that query answering in DATALOG$^\neg$ is coNP-complete in data complexity, while coNP-hardness can be easily proven by a reduction from the validity problem in propositional logic.

6 Conclusions

In this paper we have proposed methodologies and techniques for data integration and reconciliation, suitable for both the LAV and the GAV approaches. In particular, for the LAV approach we have described a method for specifying, in a declarative fashion, a set of reconciliation correspondences to be used to cope with inconsistencies among data stored in different sources. Furthermore, we have described a query rewriting algorithm that computes the set of certain answers to a user's query posed on the global schema, taking into account both the Reconciliation Correspondences and the constraints expressed over the global schema. For the GAV approach, we have shown that, when the global schema is expressed in terms of a conceptual model, the usual technique based on unfolding the query in terms of the definition of the atoms in the global schema does not guarantee completeness of the answer to a user's query. We have described a technique for query processing in GAV that overcomes this difficulty. Finally, we have defined a so-called loosely-sound semantics that supports query processing also when data at the sources are mutually incoherent, and we have proposed a technique to effectively process a query in this situation.

The techniques proposed in this paper have been developed separately for the LAV and the GAV approach. Our opinion is that the framework for data integration we have described allows one to easily adapt to the LAV setting some of the techniques we have developed for the GAV setting, and vice-versa. In particular, we believe that reconciliation correspondences can be easily adopted also in GAV, and that the loosely-sound semantics, introduced for the GAV mapping, can be generalized to the LAV mapping.

References

[1] Serge Abiteboul and Oliver Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 254–265, 1998.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[3] Foto N. Afrati, Manolis Gergatsoulis, and Theodoros Kavalieros. Answering queries using materialized views with disjunction. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 435–452. Springer, 1999.

[4] Carlo Batini, Stefano Ceri, and Shamkant B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach*. Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.

[5] Roberto Bayardo and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. of the 14th Nat. Conf. on Artificial Intelligence (AAAI'97)*, pages 203–208. AAAI Press/The MIT Press, 1997.

[6] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 99–108, 1997.

[7] Philip A. Bernstein and Thomas Bergstraesser. Meta-data support for data transformations using Microsoft Repository. *IEEE Bull. of the Technical Committee on Data Engineering*, 22(1):9–14, 1999.

[8] Alexander Borgida, Maurizio Lenzerini, and Riccardo Rosati. Description logics for data bases. In Franz Baader, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 16. Cambridge University Press, 2002. To appear.

[9] Mokrane Bouzeghoub and Maurizio Lenzerini (eds.). Special issue on data extraction, cleaning, and reconciliation. *Information Systems*, 2001.

[10] Diego Calvanese and Giuseppe De Giacomo. Expressive description logics. In Franz Baader, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 5. Cambridge University Press, 2002. To appear.

[11] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[12] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[13] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Schema and data integration methodology for DWQ. Technical Report DWQ-UNIROMA-004, DWQ Consortium, September 1998.

[14] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Source integration in data warehousing. In *Proc. of the 9th Int. Workshop on Database and Expert Systems Applications (DEXA'98)*, pages 192–197. IEEE Computer Society Press, 1998.

[15] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 194–204, 1999.

[16] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 389–398, 2000.

[17] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 58–66, 2000.

[18] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of the 15th IEEE Symp. on Logic in Computer Science (LICS 2000)*, pages 361–371, 2000.

[19] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998.

[20] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proc. of the 5th Int. Workshop on Research Issues in Data Engineering – Distributed Object Management (RIDE-DOM'95)*, pages 124–131. IEEE Computer Society Press, 1995.

[21] P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, March 1976.

[22] Sara Cohen, Werner Nutt, and Alexander Serebrenik. Rewriting aggregate queries using views. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 155–166, 1999.

[23] J. Crawford and L. Auton. Experimental results on the crossover point in random 3SAT. *Artificial Intelligence*, 81(1–2):31–57, 1996.

[24] Giuseppe De Giacomo. Intensional query answering by partial evaluation. *J. of Intelligent Information Systems*, 7(3):205–233, 1996.

[25] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 109–116, 1997.

[26] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 778–784, 1997.

[27] Thomas Eiter, Georg Gottlob, and Heikki Mannilla. Disjunctive Datalog. *ACM Trans. on Database Systems*, 22(3):364–418, 1997.

[28] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. An extensible framework for data cleaning. Technical Report 3742, INRIA, Rocquencourt, 1999.

[29] Cheng Hian Goh, Stéphane Bressan, Stuart E. Madnick, and Michael D. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Trans. on Information Systems*, 17(3):270–293, 1999.

[30] Gösta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 1999.

[31] Stéphane Grumbach, Maurizio Rafanelli, and Leonardo Tininini. Querying aggregate data. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 174–184, 1999.

[32] Jarek Gryz. Query folding with inclusion dependencies. In *Proc. of the 14th IEEE Int. Conf. on Data Engineering (ICDE'98)*, pages 126–133, 1998.

[33] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Bull. of the Technical Committee on Data Engineering*, 18(2):3–18, 1995.

[34] Alon Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 2001. To appear.

[35] Joachim Hammer, Hector Garcia-Molina, Jennifer Widom, Wilburt Labio, and Yue Zhuge. The Stanford data warehousing project. *IEEE Bull. of the Technical Committee on Data Engineering*, 18(2):41–48, 1995.

[36] Sandra Heiler, Wang-Chien Lee, and Gail Mitchell. Repository support for metadata-based legacy migration. *IEEE Bull. of the Technical Committee on Data Engineering*, 22(1):37–42, 1999.

[37] Joseph M. Hellerstein, Michael Stonebraker, and Rick Caccia. Independent, open enterprise data integration. *IEEE Bull. of the Technical Committee on Data Engineering*, 22(1):43–49, 1999.

[38] Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, 1997.

[39] Richard Hull and Gang Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 481–492, 1996.

[40] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis, editors. *Fundamentals of Data Warehouses*. Springer, 1999.

[41] Phokion G. Kolaitis and Christos H. Papadimitriou. Why not negation by fixpoint? *J. of Computer and System Sciences*, 43(1):125–144, 1991.

[42] Alon Y. Levy. Answering queries using views: A survey. Technical report, University of Washinghton, 1999.

[43] Alon Y. Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Logic Based Artificial Intelligence*. Kluwer Academic Publisher, 2000.

[44] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, pages 95–104, 1995.

[45] Chen Li and Edward Chang. Query planning with limited source capabilities. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 401–412, 2000.

[46] Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Yannis Papakonstantinou, Jeffrey D. Ullman, and Murty Valiveti. Capability based mediation in TSIMMIS. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 564–566, 1998.

[47] John W. Lloyd. *Foundations of Logic Programming (Second, Extended Edition)*. Springer, Berlin, Heidelberg, 1987.

[48] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In *Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE'95)*, pages 251–260, 1995.

[49] Xiaolei Qian. Query folding. In *Proc. of the 12th IEEE Int. Conf. on Data Engineering (ICDE'96)*, pages 48–55, 1996.

[50] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, 1995.

[51] Amit Sheth and Vipul Kashyap. So far (schematically) yet so near (semantically). In *Proc. of the IFIP DS-5 Conf. on Semantics of Interoperable Database Systems*. Elsevier Science Publishers (North-Holland), Amsterdam, 1992.

[52] D. Srivastava, S. Dar, H. V. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 318–329, 1996.

[53] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1997.

[54] Ron van der Meyden. Logical approaches to incomplete information. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publisher, 1998.

[55] Jennifer Widom (ed.). Special issue on materialized views and data warehousing. *IEEE Bull. of the Technical Committee on Data Engineering*, 18(2), 1995.

[56] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[57] Hantao Zhang. SATO: an efficient propositional prover. In *Proc. of the 14th Int. Conf. on Automated Deduction (CADE'97)*, Lecture Notes in Computer Science. Springer, 1997.

[58] Gang Zhou, Richard Hull, and Roger King. Generating data integration mediators that use materializations. *J. of Intelligent Information Systems*, 6:199–221, 1996.

[59] Gang Zhou, Richard Hull, Roger King, and Jean-Claude Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95)*, pages 4–18, 1995.