Finding Successful Queries in a Mediator Context

Alain Bidault, Christine Froidevaux and Brigitte Safar
L.R.I., C.N.R.S & Université Paris-Sud Bâtiment 490,
91405 Orsay Cedex, France

Abstract

In this paper, we study *failing* queries posed to a mediator in an information integration system and expressed in the logical formalism of the information integration system PICSEL[1] . First, we present the notion of concept generalisation in a concept hierarchy that is used to repair failing queries. Then, we address two problems arising while rewriting a query using views. The first problem concerns queries that cannot be rewritten due to a lack of sources, the second one concerns queries that have only unsatisfiable rewritings.

1 Introduction

Data integration is a problem that has recently received significant attention due to the increasing number of structured and heterogeneous sources. Users and applications today require an integrated access to multiple heterogeneous information systems and many projects (e.g., Information Manifold [15], PICSEL [13], SIMS [1], TSIMMIS [7]) have been proposed. They all provide, in a *mediator* architecture, a uniform query interface to multiple autonomous data sources, located on the world wide web for example. Users pose queries in terms of a mediated schema, that is, in terms of a set of relations designed to capture the semantics of a given application domain (e.g., tourism[2]). Those relations are *virtual* in the sense that their instances are not directly available but stored in the sources. As a consequence, answering a query means translating a user's query into a query that refers directly to the relevant sources, which needs a set of *source descriptions*. In our setting, sources are described by a set of *views*, for which logical constraints and a logical mapping with domain relations are specified.

The most important advantage of a mediator is that it enables users to focus on specifying their demand, by freeing them from having to find the relevant sources and possibly combine data from multiple sources to obtain answers. Instead, the mediator takes control of the construction of the specialised query plans (expressed in terms of views) to be executed in order to answer the original queries (expressed in terms of the domain model).

[1]PICSEL is a project supported by the CNET (Centre National d'Études des Télécommunications) under contract number 97 1B 378.

[2]PICSEL is in collaboration with the travel agency Degriftour, http://www.degriftour.fr/

In the setting of data integration systems, the need for a *cooperative* query answering process is especially crucial because users do not know the contents of the data sources that are available.

Many works have been done in the field of cooperative answering ([10],[17],[4],[6]), particularly on the problem of failing queries i.e. queries having an evaluation that produces an empty answer set. In that situation, Godfrey [14] says that a query Q may fail because one of its subqueries is evaluated to false in DB, DB $\not\models \exists \bar{X}.Q(\bar{X})$: he says that it contains a *false presupposition.* Furthermore, if one of its subqueries leads to contradiction because of the integrity constraints associated to the database, DB $\models \neg\exists \bar{X}.Q(\bar{X})$, he says that it also contains a *misconception*. Godfrey proposes to report to the user, in the general case, a *minimal failing sub-query* (MFS), and in the second case, a *minimal conflicting sub-query* (MCS). In both cases, he shows that looking for all the problematic sub-queries and finding all their repairs are NP-hard problems, which can become polynomial when exhaustiveness is not required. Let us remark that MFS cannot be detected without effective access to the data of the sources while MCS can.

Gal [11] confronts the user's query with integrity constraints before consulting the database in order to give the user the integrity constraints that have been violated. As she does not offer any repair, she need not find the MCS.

Motro [18], [19] performs the MFS by testing all the immediate generalisations of the initial query and by iterating this process on all the failing generalisations. The MFS are those for which all generalisations succeed. A query generalisation is done by deleting one of its literals or by varying one of its parameters along a range of predefined values. Motro does not have to detect all the conflicts beforehand, but, instead, he must often consult the effective data of the sources.

Gaasterland [8][9] modifies the query either to get more information, which is relevant for the user, or to repair the query if it fails. She describes how to relaxe each predicate or constant arguments. As there are many possibilities, relaxations are generated in breadth first, and at each level, they are submitted to the user, who has to decide, interactively, which relaxation he prefers. If the query fails, Gaasterland re-uses Gal's work to propose, first, the relaxations that repair the query.

Our objective is to help a user to reformulate his failing query, without effective access to the contents of the sources. In a previous paper [2], we have considered the problem of repairing queries which do not obtain any answer, due to a violation of constraints. We have shown how to identify all the MCS and to construct minimal repairs, i.e. queries that generalise the user's query and that do not violate any domain constraint.

In this paper, we consider that the user's query, while being meaningful w.r.t the domain model, has no answer because its rewritings lead to specialised query plans that violate the constraints specifying the actual contents of the sources. For instance, the user asks for hotels located in England, and the only sources connected to the mediator provide hotels located in Germany. It is very useful to offer him a new query, called a *repair*, which is semantically close to the initial one and for which the mediator could provide answers. For instance, the user could be interested in a source providing Bed&Breakfasts in England, instead of hotels.

Also, we take a logical framework (see section 2) for representing the domain model and the source description, associated with inference algorithms. These algorithms are the basis for computing the specialised query plans, and for checking their compatibility w.r.t the constraints. In section 3, we introduce, using an example, the notion of concept generalisation in a concept hierarchy, and the notion of repairs for failing queries. In section 4, we address the problems arising while rewriting a query using views. The first problem concerns queries that cannot be rewritten due to a lack of sources. The second one concerns queries which have only unsatisfiable rewritings.

2 Representation of Domain, Sources and Queries

First, we specify the logical framework, called the domain theory, of our study. We describe the formalism used to express knowledge in the application domain, the contents of the sources relative to the domain and queries.

We place ourselves in the perspective of extending our framework such that it captures the expressiveness of the CARIN language. This language combines a description logic with Horn rules (without function symbols). It assumes that the terminological component completely describes the hierarchical structure in the domain and therefore, rules do not allow to make new inferences on this structure [16]. Here, our rules are not the intensional rules found in the domain of deductive databases [5].

2.1 Domain Knowledge

The knowledge domain is expressed by means of a declarative representation of object classes (Country, Flight, Stay, Travel...) and of relations among these classes. The domain is described using atoms, of the form $p(\bar{X})$ where p is a relation name and $\bar{X}$ a tuple of variables or constants. We distinguish some unary relations, called *concepts*. They represent object classes relevant to the application domain. $C(x)$ is called an *atom-concept* if C is a concept.

The domain knowledge $(\mathcal{D}, \mathcal{C})$ contains two components:

- **A set** $\mathcal{D}$, composed by safe rules of the form: $p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow q(\bar{Y})$, where $p_1(\bar{X}_1) \ldots p_n(\bar{X}_n)$ and $q(\bar{Y})$ are atoms, $\bar{X}_1, \ldots, \bar{X}_n$ and $\bar{Y}$ are tuples of variables, and where each variable in $\bar{Y}$ appears in $\bar{X}_1 \cup \ldots \cup \bar{X}_n$.

$\mathcal{D}_h$ is the set of particular rules that describes a hierarchy between the domain concepts, and that are of the form $C_1(x) \Rightarrow C_2(x)$, where C_1 and C_2 are concepts.

ex: $r_1 \in \mathcal{D}$: Flight(x) $\wedge$ Stop(x, y) $\wedge$ Berlin(y) $\Rightarrow$ FlightViaBerlin(x) (a flight via Berlin is a flight having a stop in Berlin). $r_2 \in \mathcal{D}_h$: DirectFlight(X) $\Rightarrow$ Flight(X), (a direct flight is a flight).

Concepts and relations not appearing in rules as conclusions, except in $\mathcal{D}_h$, are called *base relations*.

- **A set** $\mathcal{C}$, composed by Horn constraints of the form: $c : l_1(\bar{X}_1) \wedge \ldots \wedge$

$l_n(\bar{X}_n) \Rightarrow \perp$ where $l_1(\bar{X}_1), \ldots, l_n(\bar{X}_n)$ are **literals.**

ex: A place in Berlin is not in Paris: Berlin(x) $\wedge$ Paris(x) $\Rightarrow \perp$

2.2 Description of the Sources

The contents of a source $\mathcal{S}_i$ are represented using a vocabulary $\mathcal{V}$ constituted by as many local relations V_{ij}, called *views*, as we know that the source $\mathcal{S}_i$ gives instances of the domain base relations. The description of sources in terms of views contains two components:

- **A logical set of implications** $\mathcal{D}_v$**,** $V_{ij}(\bar{X}) \Rightarrow p(\bar{X})$ that links each view to a domain relation.

ex: the source $\mathcal{S}_1$ provides instances of the concept *Hotel* linked to instances of the binary relation *Located.* $V_{11}(x) \Rightarrow$ Hotel(x), $V_{12}(x, y) \Rightarrow$ Located(x, y)

- **A set of Horn constraints** $\mathcal{C}_v$ characterising the view instances: $l_1(\bar{X}_1) \wedge \ldots \wedge l_n(\bar{X}_n) \Rightarrow \perp$, where $l_1 \ldots l_n$ are base relations and/or view names or their negation. **ex:** hotels from $\mathcal{S}_1$ are located in the Caribbean, and instances from $V_{12}(x, y)$ are linked to instances from $V_{11}(x)$:

$V_{11}(x) \wedge Located(x, y) \wedge \neg InCaribbean(y) \Rightarrow \perp \qquad V_{12}(x, y) \wedge \neg V_{11}(x) \Rightarrow \perp$

The Semantics used is standard first order logic semantics.

Note that, in our approach, the terminological component is composed of the two sets $\mathcal{D}_h$ and $\mathcal{D}_v$.

We define the domain theory $\mathcal{T} = \mathcal{D} \cup \mathcal{D}_{\mathcal{V}} \cup \mathcal{C} \cup \mathcal{C}_{\mathcal{V}}$. Assume that $\mathcal{T} \not\models \perp$, with the following property of coherence: for any relation p of the domain, $\mathcal{T} \not\models \forall \bar{X} \neg p(\bar{X})$, and that the dependency graph of the relations appearing in $\mathcal{T}$ is acyclic.

Note that our assumption about non recursivity is restrictive but motivated by a need of efficiency while answering queries in the general framework of CARIN. As rules are not recursive, they can be unfolded i.e. converted into an equivalent set of rules. Using efficient algorithms is the keystone of the CARIN language.

2.3 Position of the Problem

We perform conjunctive queries of the form: $Q(\bar{X}) : p_1(\bar{X}_1, \bar{Y}_1, \bar{a}_1) \wedge \ldots \wedge p_n(\bar{X}_n, \bar{Y}_n, \bar{a}_n)$, where p_i are domain relation names appearing in $\mathcal{D} \cup \mathcal{C}$. The variables $\bar{X} = \bigcup_{i=1}^{n} X_i$ are called *distinguished variables* of the query and represent data, instances expected by the user when setting the query. $\bar{Y} = \bigcup_{i=1}^{n} Y_i$ are non-distinguished variables, and $\bar{a}_1, \ldots, \bar{a}_n$ are tuples of constants. The variables are existentially quantified.

Classically, answering a query $Q(\bar{X})$ is interpreted relatively to a database $\mathcal{DB}$, possibly associated to a domain theory $\mathcal{D}$, and consists in determining whether:

$\mathcal{DB}, \mathcal{D} \models \exists \bar{X}, \exists \bar{Y} [p_1(\bar{X}_1, \bar{Y}_1, \bar{a}_1) \wedge \ldots \wedge p_n(\bar{X}_n, \bar{Y}_n, \bar{a}_n)]$.

In the case of a positive answer, the *answer* to the query is the set of tuples $\bar{b} = \bar{b}_1 \cup \ldots \cup \bar{b}_n$ such that: $\mathcal{DB}, \mathcal{D} \models \exists \bar{Y} [p_1(\bar{b}_1, \bar{Y}_1, \bar{a}_1) \wedge \ldots \wedge p_n(\bar{b}_n, \bar{Y}_n, \bar{a}_n)]$

In fact, in our mediator approach, we do not access to the data source contents. Answering a query consists in searching the different expansions of this query in terms of views. An *expansion* of a query $Q(\bar{X})$ is a query that, using the domain theory $\mathcal{T}$, logically implies $Q(\bar{X})$, and that is satisfiable with $\mathcal{T}$.

In the PICSEL [13] mediator context, an algorithm that determines expansions in terms of views has been developed and implemented in Java. This algorithm proceeds, using $\mathcal{D}$ and $\mathcal{D}_{\mathcal{V}}$, by successive rewritings $Q_{\mathcal{R}}(\bar{X})$ of the initial query $Q(\bar{X})$ in a backward chaining way with a saturation phase using $\mathcal{C}_{\mathcal{V}}$. We have $Q_{\mathcal{R}}(\bar{X}), \mathcal{D} \cup \mathcal{D}_{\mathcal{V}} \cup \mathcal{C}_{\mathcal{V}} \models Q(\bar{X})$.

These rewritings are the nodes of a tree rooted by Q. While developing the tree, for each node $Q_{\mathcal{R}}$, the satisfiability with $\mathcal{T}$ is tested. A node is a leaf either if $Q_{\mathcal{R}}(\bar{X}), \mathcal{T} \models \perp$, or if $Q_{\mathcal{R}}$ can no longer be rewritten.

Definition 2.1: *A* **terminal rewriting** *is a rewriting of a query where each atom has been rewritten in terms of views.*

Definition 2.2: *A* **successful query** *is a query that has at least a satisfiable terminal rewriting. Otherwise, it is a* **failing query**.

This mechanism used to expand the query bears some analogy with the one presented by [5] and by Pirotte et al. in [20] to build intensional answers. In PICSEL all the satisfiable terminal rewritings, and only them, have to be evaluated against the sources, in order to be sound and complete.

Then, a problem arises when all the leaves are unsatisfiable with $\mathcal{T}$, or contain at least an atom that is not a view. Our aim is to obtain repairs of $Q(\bar{X})$, that are the least successful generalisations of $Q(\bar{X})$.

In [2], working on unsatisfiable queries $Q(\bar{X})$ i.e. such that $Q(\bar{X}), \mathcal{T} \models \perp$, we have shown how to identify the atoms of the query responsible of the unsatisfiability, and we have defined an algorithm that generalises these atoms in order to obtain a satisfiable query. In this paper, we will assume that $Q(\bar{X})$ is satisfiable: $Q(\bar{X}), \mathcal{T} \not\models \perp$, but is failing as each of its rewritings is unsatisfiable. Our problem is to check if the techniques used in [2], presented below through an example, are re-usable.

3 Concept Generalisation and Query Repairs

Example 1: Figure 1 presents a fragment of the concept hierarchy $\mathcal{D}_h$. Assume that we have the following two sources which offer hotels located in the Sahara for the first one, and in Reunion for the second one.

S1: $\mathcal{D}_{v1}$	r_{v11}	$V_{11}(x) \Rightarrow Hotel(x)$	r_{v12}	$V_{12}(x,y) \Rightarrow Located(x,y)$
$\mathcal{C}_{v1}$	c_{v11}	$V_{12}(x,y) \wedge \neg V_{11}(x) \Rightarrow \perp$		
	c_{v12}	$V_{11}(x) \wedge Located(x,y) \wedge \neg InSahara(y) \Rightarrow \perp$		
S2: $\mathcal{D}_{v2}$	r_{v21}	$V_{21}(x) \Rightarrow Hotel(x)$	r_{v22}	$V_{22}(x,y) \Rightarrow Located(x,y)$
$\mathcal{C}_{v2}$	c_{v21}	$V_{22}(x,y) \wedge \neg V_{21}(x) \Rightarrow \perp$		
	c_{v22}	$V_{21}(x) \wedge Located(x,y) \wedge \neg Reunion(y) \Rightarrow \perp$		

$\mathcal{C}$	$c_1:$	$InSahara(x) \wedge PlaceWithSea(x) \Rightarrow \perp$
	$c_2:$	$NoBeachPlace(x) \wedge IslandWithBeach(x) \Rightarrow \perp$

Suppose that the user wishes to get an hotel in Madeira:

Q(x) = Hotel(x) ∧ Located(x, y) ∧ Madeira(y).

In order to optimize the identification of unsatisfiable rewritings, the mediator always begins by rewriting the n-ary relations which appear in constraints. In its first expansion path, working on $Located(x, y)$ and using r_{v12} and r_{v22} in the backward chaining, it constructs the two following rewriting nodes:
$Q_{\mathcal{R}_1}$: $Hotel(x) \wedge V_{12}(x, y) \wedge Madeira(y)$ $\quad$ $Q_{\mathcal{R}_2}$: $Hotel(x) \wedge V_{22}(x, y) \wedge Madeira(y)$.
Because these two rewritings are unsatisfiable with $\mathcal{T}$, the algorithm stops the expansion process.

First, we must identify, for each rewriting, the atoms of the initial query responsible of the unsatisfiability. The saturation of $Q_{\mathcal{R}_1}$ with $\mathcal{T}$ shows that the atom

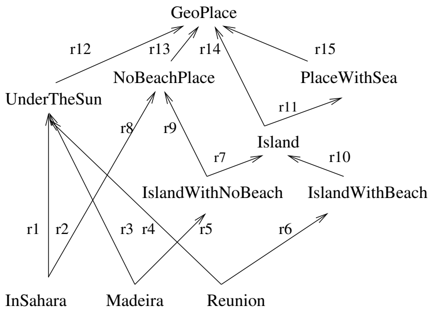

Figure 1: Geographical Places Hierarchy

$Madeira(y)$ from which $IslandWithNoBeach(y)$, $Island(y)$ and $PlaceWithSea(y)$ are derived using r_5, r_7 and r_{11}, is the source of the violation of the constraint c_1: the knowledge used for the derivation of $\perp$ is $V_{12}(x, y), Madeira(y)$, and r_{v12}, r_5, r_7, r_{11}, c_{v11}, c_{v12}, c_1. In $Q_{\mathcal{R}_2}$, $Madeira(y)$ is also responsible of the unsatisfiability, because of c_2 and of the derivation of $NoBeachPlace(y)$ by r_5 and r_9.
The knowledge used for the derivation of $\perp$ is $V_{22}(x, y), Madeira(y)$, and r_{v22}, r_5, r_6, r_9, c_{v21}, c_{v22}, c_2.

In [2] we showed that we can repair a query generalising the atoms C responsible of the inconsistency, taking into account the rules r_h of $\mathcal{D}_h$ used in the derivation of $\perp$ and rooted in these atoms.

Let us begin by defining the direct generalisation of an atom-concept $C(x)$.

Definition 3.1: *Let $\mathcal{C}$ be a concept and $\mathcal{C}_1$, $\mathcal{C}_2$,..., $\mathcal{C}_n$, concepts such that the rules $\mathcal{C}(x) \Rightarrow \mathcal{C}_i(x)$, with $1 \leq i \leq n$, are in $\mathcal{D}_h$, the* **direct generalisation** *$\mathcal{G}_c(x)$ of the* **atom-concept** *$\mathcal{C}(x)$ is the conjunction $\mathcal{C}_1(x) \wedge \mathcal{C}_2(x) \wedge ... \wedge \mathcal{C}_n(x)$.*

ex: $\mathcal{G}_{Madeira}(y)$ is $UnderTheSun(y) \wedge IslandWithNoBeach(y)$

Remarks: (i) We use the conjunction of the $\mathcal{C}_i(x)$ to stay close to the concept, taking into account all its features. (ii) We have the property: $\mathcal{C}(x), \mathcal{D}_h \models \mathcal{G}_c(x)$.

Definition 3.2: *The* **over-step generalisation** *$\mathcal{G}_{c-r_h}(x)$ is the direct generalisation* *of the atom-concept $\mathcal{C}(x)$ where all the concepts reached by applying rules of r_h have been successively replaced by their direct generalisations.*

ex: The over-step generalisations obtained are:
for $Madeira(y)$-$\{r_5, r_7, r_{11}\}$, $UnderTheSun(y) \wedge NoBeachPlace(y) \wedge GeoPlace(y)$, and for $Madeira(y)$-$\{r_5, r_9\}$, $UnderTheSun(y) \wedge Island(y) \wedge GeoPlace(y)$.[3]

Because queries are conjunctive, we have to define the generalisation of a conjunction of atom-concepts.

Definition 3.3: *A* **direct generalisation** $\mathcal{G}_d(x)$ *of a* **conjunction of atom-concepts** $AC_c = \mathcal{C}_1(x) \wedge \ldots \wedge \mathcal{C}_{i-1}(x) \wedge \mathcal{C}_i(x) \wedge \mathcal{C}_{i+1}(x) \wedge \ldots \wedge \mathcal{C}_n(x)$ *is a conjunction of the form* $\mathcal{C}_1(x) \wedge \ldots \wedge \mathcal{C}_{i-1}(x) \wedge \mathcal{G}_{ci}(x) \wedge \mathcal{C}_{i+1}(x) \wedge \ldots \wedge \mathcal{C}_n(x), 1 \leq i \leq n$, *where* $\mathcal{G}_{ci}(x)$ *is the direct generalisation of* $\mathcal{C}_i(x)$. **Generalisations** *of* AC_c *are obtained using the transitive closure of the direct generalisation relation.*

Definition 3.4: *Let* $Q(\bar{X}) = \mathcal{C}_{ac}(x) \wedge q(\bar{X})$ *be a failing query, where* $\mathcal{C}_{ac}(x)$ *is a conjunction of atom-concepts, and* $q(\bar{X})$ *a conjunction of literals with* $x \in \bar{X}$. $\mathcal{R}(\bar{X}) = \mathcal{C}'_{ac}(x) \wedge q(\bar{X})$ *is a* **repair** *of* $Q(\bar{X})$ *if* $\mathcal{C}'_{ac}(x)$ *is a generalisation of* $\mathcal{C}_{ac}(x)$ *and if* $\mathcal{R}(\bar{X})$ *is a successful query.*

Many repairs can satisfy the previous definitions, and only minimal repairs have to be given to the user.

Definition 3.5: $\mathcal{R}(\bar{X})$ *is a* **minimal repair** *of the failing query* $Q(\bar{X})$ *if* $\mathcal{R}(\bar{X})$ *is a repair of* $Q(\bar{X})$ *and if there does not exist* $\mathcal{R}_2(\bar{X})$, *a repair of* $Q(\bar{X})$, *such that* $\mathcal{R}_2(\bar{X}), \mathcal{D}_h \models \mathcal{R}(\bar{X})$.

In our example, because our domain is acyclic, we can construct, using the over-step generalisation on each rewriting, two repairs of the initial query: $\mathcal{R}_1$: Hotel(x) $\wedge$ Located(x, y) $\wedge$ UnderTheSun(y) $\wedge$ NoBeachPlace(y), associated with the rewriting $Q_{\mathcal{R}_1}$ and $\mathcal{R}_2$: Hotel(x) $\wedge$ Located(x, y) $\wedge$ UnderTheSun(y) $\wedge$ Island(y), with $Q_{\mathcal{R}_2}$.

4 Problems arising because of the sources

In this example, our approach seems to be reasonable. However, there are several problems. First, having only two sources reduces the number of rewritings and, fortunately, the two generalisations found for $Madeira(y)$ are compatible with these sources. In fact, many sources are reachable: even if they only rewrite *one* atom of the query, they can produce unsatisfiable rewritings that will have to be taken into account.

Second, knowing each cause of inconsistency in each rewriting is expensive.

Third, when we generalise a rewriting, we do not know necessarily if there exist other sources that specialise the literals added by generalisation and permit to obtain a satisfiable rewriting: only PICSEL can prove that a *potential* repair has a satisfiable terminal rewriting. At last, using our method on specialisations of the initial query

[3]GeoPlace(y) will be removed because it is a generalisation of UnderTheSun(y).

can produce repairs that are not minimal. We illustrate the problem with an example.

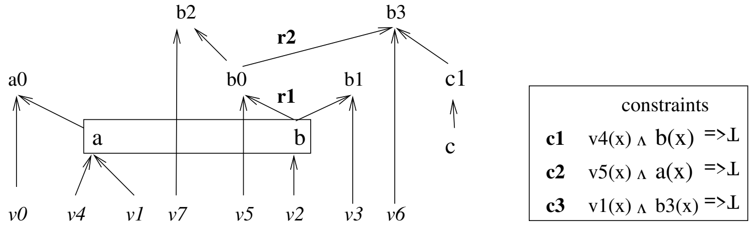

Figure 2: Views reachable from concepts of the hierarchy

Example 2: We use the knowledge base shown in figure 2. The query is: Q(x)= $a(x) \wedge b(x)$. The two rewritings $v4(x)$ and $v1(x)$ of $a(x)$ are incompatible with $b(x)$, because of c_1 and c_3. Our techniques propose three generalisations: to over-step $a(x)$-$\emptyset$, $Q_{\mathcal{R}_1} = a_0(x) \wedge b(x)$, to over-step $b(x)$-$\emptyset$, $Q_{\mathcal{R}_2} = a(x) \wedge b_0(x) \wedge b_1(x)$. and to over-step $b(x)$-r_1, r_2, $Q_{\mathcal{R}_3} = a(x) \wedge b_1(x) \wedge b_2(x)$[4].

Therefore, we see on figure 2 that $Q_{\mathcal{R}_2}$ is not a repair: if $b(x)$ is replaced by $b0(x) \wedge b1(x)$, two more views, $v5(x)$ and $v3(x)$, are reachable, but they do not permit to obtain a satisfiable rewriting because of c_2. Moreover, $Q_{\mathcal{R}_3}$ is not minimal: deleting $b_3(x)$ is not necessary; the query $a(x) \wedge b_1(x) \wedge b_2(x) \wedge b_3(x)$ admits the following satisfiable rewriting $v4(x) \wedge v3(x) \wedge v7(x) \wedge v6(x)$.

Therefore, the techniques introduced in [2] cannot be directly applied ; other directions have to be investigated. In the following, we propose a general issue for our problem without exploiting the knowledge about the causes of unsatisfiability. Then, we show that we could use this knowledge in order to get repairs in a less expensive way.

4.1 Pre-treatment.

In order to be sure that any atom of the query can be rewritten in at least one way, in terms of views, in a backward chaining process, we propose a cooperative pre-treatment of the conjunction.

Definition 4.1: *A* **pertinent atom** *is an atom that can be rewritten in terms of views, using $\mathcal{D} \cup \mathcal{D}_v \cup \mathcal{C}_v$.*

Thus, each atom of the query that does not verify this property must be replaced by its direct generalisation and so on to obtain a conjunction of pertinent atoms.

Example 1 continued: According to the sources, Madeira(x) is not pertinent ; it should be replaced in a query by its generalisation that contains only pertinent atoms, that is $UnderTheSun(x) \wedge NoBeachPlace(x) \wedge Island(x)$.

[4]As $b_3(x)$ cannot be generalised, it is simply removed

To know whether an atom is pertinent, we store in a table the atoms already detected as pertinent, so that we only have to test the atoms not appearing in this table. When sources are added or deleted, the table has to be updated.

Moreover, it is useless to propose a generalisation to the user if we already know for sure that it will not get any results. Thus, these generalisation steps must be applied to our potential repairs too: there is no need to choose as a potential repair a generalisation if its only rewritings are those we already get before having generalised the previous conjunction.

In the following, we assume that all the queries submitted to PICSEL are only constituted by pertinent atoms. Unfortunately, it does not mean that they will be successful, as inconsistencies can arise with the rewritings.

4.2 Completeness, minimality and user's preferences.

As there are many sources accessible from the mediator, and many different generalisations of a query, it is not reasonable to expect, for a concrete application, to be complete by proposing ***all*** the minimal repairs of the query. However we can propose minimal repairs that respect the user's preferences.

In [19], Motro shows a query generalisation method using a lattice over query generalisations; we have adapted it in order to take into account our hierarchy over concepts and a preference order over literals given by the user[5].

If we generalise an atom, we can hope that new sources will be reachable and will make up a satisfiable rewriting. We cannot guess it before asking PICSEL . We have to iterate this process over the potential repairs we obtain. To get a total order over atoms, given a hierarchy tree, we decide that literals *on the left* are less important than literals placed *on the right*. Obviously, as soon as a repair is found, none of its generalisations is investigated.

Example 2 continued: We have seen above that the query: Q(x)= $a(x) \wedge b(x)$ is unsatisfiable. PICSEL tells us that $b0(x) \wedge b1(x) \wedge a(x)$ is not successful ; thus, to obtain a repair, we have to iterate, generalising first $b0(x)$.

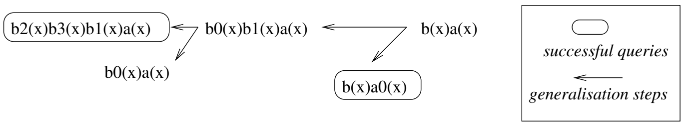

Figure 3: The potential repairs obtained

This method returns all the minimal repairs of the query, but only them. Furthermore, the first repair we get, $b_2(x) \wedge b_3(x) \wedge b_1(x) \wedge a(x)$, is the one that fits the user's preferences. The presence of many sources has two effects: it does not encourage us to be exhaustive, it also means that we can hope that a few steps of generalisation are needed in order to obtain answers.

[5]By default, the last atom is the least important, that is the first to generalise.

4.3 Restrictions for more efficiency.

Assume now that the user accepts to obtain repairs that do not exactly follow his preferences, but wants to get a repair close to his query, that is not too general. Then, the rewritings generated by PICSEL have useful information about which atoms to relax in the query. These rewritings should be sorted according to:

- The number of unrewritten atoms. By looking for a query entirely rewritten, we can hope that many close sources are available. The repair of such a rewriting should be obtained applying a few generalisation steps.

- The number of rules from the hierarchy $\mathcal{D}_h$ used in the backward chaining process. Every time a rule is used, it means that an atom from the query has been specialised: UnitedKingdom(x)$\Rightarrow$ IslandWithBeach(x) $\Rightarrow$ Island(x).

A repair too much specialised is not satisfactory, because the answers obtained from its rewritings could be very far away from the answers the user expected. For example, if the user asks for *an hotel on an Island*, and if his query is failing, it can be rewritten with the terms of a source that provides *campsites in the United Kingdom.*

-The user's preferences.

Then, once we have sorted in a list all the rewritings according to these three criteria, we apply the following algorithm:

repeat (see 4.2)

(i) detect the atoms at the source of the inconsistency for the first rewriting of the list, using, for example, the method we defined in [2] based on a saturation of the rewritings with $\mathcal{T}$,

(ii) identify the atoms from the query that are linked to the rewriting views,

(iii) generalise the least important atom (following the user's preferences).

until one or all the repairs of the query are found.

The first repair we find is minimal. As we were guided by the literals of an unsatisfiable rewriting, we should find faster a successful repair. But, we just have partially followed the user's preferences to choose the atoms to generalise.

At least, if we do not want to spend time to propose a repair, we should have already identified queries, learnt or pre-defined, that we know they succeed. We will choose among them the closest to the user's query.

5 Conclusion and Perspectives

We place ourselves in a mediator context where a hierarchy of concepts is used to define the terminological component. We assume that the user sets a query that cannot be fully rewritten in terms of views. Our objective is to help the user to re-formulate his failing query. This paper has introduced the notion of minimal repairs of a query by means of concept generalisations. The failures encountered, in this context, have been listed and we have shown that looking after every minimal repair is an expensive process. Nevertheless, methods to propose successful repairs have been presented. Their cost depends on whether we (or the user) want to find a minimal repair or just any repair.

Note that in our formalism, we do not use functional symbols and that we use sometimes predicate symbols rather than constant symbols. Indeed, in order to be able to replace an object by a more general one (e.g. Madeira by Island), Chu and Chen, in [6], propose a type abstraction hierarchy and use an abstract mapping with tables to link together instances represented at different knowledge levels to concepts. We prefered to consider this object as a concept in a hierarchy. This approach is commonly used in the context of Inductive Logic Programing (ILP) and corresponds to the flattening transformation presented in [21]. Moreover, successful queries could be considered as positive examples and integrity constraints as negative examples ([22]). It would be interesting to integrate some tools of ILP's approach in our methods to solve the problem of query repairing.

The work we have presented here is meant to be seen as a first study of repairing queries in a mediator architecture where knowledge is expressed by Horn rules. We plan to extend it in order to take into account the full expressiveness of the formalism CARIN [16], which combines Horn rules with description logics.

Finally, optimisations have to be found to improve the cooperative behavior, so that the PICSEL mediator becomes a very powerful tool. Thus, we have to (i) conceive new methods to order and group the rewritings in order to find the best starting point for the algorithm exposed in 4.3, and (ii) propose minimal repairs that are close to the user's preferences, without having to calculate an entire lattice as we shown in 4.2.

References

[1] Y. Arens, C. A. Knoblock and W.-M. Shen. Query reformulation for dynamic information integration. *J. Intelligent Information System*, 6, 96.

[2] A. Bidault, C. Froidevaux and B. Safar. Repairing Queries in a Mediator Approach. *ECAI 2000, Berlin*.

[3] Chang and Lee. Symbolic Logic and Mechanical Theorem Proving, 100-121, Academic Press, 73.

[4] L. Cholvy Answering queries addressed to a group of deductive databases. In *Workshop on Flexible Query-Answering Systems*, Danemark, May 96.

[5] L. Cholvy and R. Demolombe Querying a Rule Base. In *Expert Database Systems*, Larry Kerschberg Editor, 477-485, 1987.

[6] W. Chu and Q. Chen. A Structured Approach for Cooperative Query Answering. *IEEE Transactions on Knowledge and Data Engineering,* 6(5):738-749, October 94.

[7] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogenous information sources. In *Proceedings of IPSJ*, Tokyo, October 94.

[8] T. Gaasterland, P. Godfrey and J. Minker. Relaxation as a Platform of Cooperative Answering. *Journal of Intelligent Information Systems*, Vol. 1, No: 3/4, 293-321, December 92.

[9] T. Gaasterland. Cooperative Answering through Controlled Query Relaxation. *IEEE Expert*, 48-59, Sept-Oct 97.

[10] T. Gaasterland, P. Godfrey, and J. Minker. An overview of Cooperative Answering. *Journal of Intelligent Information Systems*, (1) 123-157, 92.

[11] A. Gal. Cooperative Responses in Deductive Databases. *PhD thesis, Departement of Computer Science, University of Maryland*, 88.

[12] M. R. Genesereth, A. M. Keller, and O. M. Duschka. Infomaster: An information integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD-97*, 97.

[13] F. Goasdoué, V. Lattes and M.-C. Rousset. The Use of CARIN Language and Algorithms for Information Integration: The PICSEL Project. In *Int. J. of Cooperative Information Syst. 99.*

[14] P. Godfrey. Minimization in cooperative response to failing databases queries. *Int. J. of Cooperative Information Syst.* 6(2): 95-149, 97.

[15] A. Levy, A. Rajamaran, J. Ordille. Query-answering algorithms for information agents. *Proc AAAI'96*, 40-47.

[16] A. Levy and M.-C. Rousset. Combining Horn Rules and Description Logics in CARIN. *Artificial Intelligence*, 104, 165-209, 98.

[17] J. Minker. An overview of Cooperative Answering in Databases In *Proc FQAS'98*, 282-285, May 98.

[18] A. Motro. Intensional Answers to Database Queries. *IEEE*, 6(3), 444-454, June 94.

[19] A. Motro. Flex: A tolerant and cooperative user interface to databases. *IEEE Transactions on Knowledge and Data Engineering,* 2(2):231-246, June 90.

[20] A. Pirotte, Dominique Roelants and Esteban Zimanyi Controlled Generation of Intensional Answers *IEEE Transactions on Knowledge and Data Engineering,* 3(2):221-236, June 91.

[21] C. Rouveirol. Flattering and Saturation: Two Representation Changes for Generalization. *Machine Learning*, 14, 219-232, 94.

[22] M. Sebag and C. Rouveirol. Induction of Maximally General Clauses Consistent with Integrity Constraints. *Proc 4th Int. Workshop on Inductive Logic Programming*, 195-215, ILP-94.