National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

SAFIR, Analogy-based System for Database Definition and Query Reuse

by

Hamid Ould-Brahim

Thesis
submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master in Computer Science

The Ottawa-Carleton Institute for Computer Science
University of Ottawa

ISBN 0-315-89705-8

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Acknowledgments

I would like to express my deepest appreciation to my thesis supervisor Professor Stan Matwin for his guidance, encouragement, and advice throughout the course of my study and research. I have also to mention that he was helpful to me both on and off the campus. Many thanks also go to Dr. Robert Holte for his invaluable suggestions and comments during the progress of my work.

My deepest respect goes to my parents who gave me the moral support and encouragement to further my education in abroad. I am indebted to my wife Nabila for all the support, the understanding and patience she shows throughout all my graduate studies.

I would like to thank all the members of the Ottawa Machine Learning Group, particularly the Software Reuse Group for the fruitful discussions we had during the research and the writing of this thesis.

Finally, I gratefully acknowledge the financial assistance provided by the Algerian Government, Natural Science and Engineering Research Council (Canada) and the University of Ottawa.

Abstract

This thesis presents a system which employs analogical reasoning to define and manipulate databases. The objective is to model the behavior of naive users who often adapt database relations and queries from documentation examples to the problem at hand. The user has at her disposal a manual that contains examples of database designs with a variety of database queries in some exemplary milieu called the *base domain*. The manual explains the example queries in simple terms and gives their implementation in a database query language such as SQL. Our user is not a database expert. She can neither define the database nor can she compose query implementations from scratch. At best she can construct simple queries in the base domain by copying them verbatim from the manual. Our system works in two steps: first it defines a target database by building incrementally analogies represented by pairs of object correspondences, base-target. These analogies are built by mapping objects, relationships, and semantic constructs from the base domain into the target domain. The process is then interactive and incremental (i.e. the user can accept or reject the inferred analogies). The second step of the approach consists to derive target domain query's implementation from its specification by reusing queries developed in the base domain. We rely on a weak form of derivational analogy—traces of a planner operating in the base domain are used to derive a target domain query's implementation in SQL from its specification. The very simple planner used builds in the base domain plans that result in particularly simple, prototypical queries. Queries are specified in a high level subset of natural language.

Outline

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

This thesis presents a system dubbed SAFIR (Analogy-based System for DAtabase deFInition and Query Reuse) which employs analogical reasoning to define and manipulate databases [Ould-Brahim and Matwin 1992]. Our objective is to model the behavior of a naive user of a database system. We tackle here the main problems that a typical naive database user encounters when developing from scratch her own application. Indeed, the first problem facing the user is to define her database according to an existing conceptual model [Date 1990]. This means that the user must follow some database design techniques in order to build the relations making up the database. In practice, these relations are derived from a conceptual model such as the entity-relationship model or from formal methods like the normalization approach. Therefore, a database user must know *how* to use one of these techniques prior to any effective use of the database system.

Another problem facing the novice database user is her ability to formulate correct queries using an existing query language such as SQL, QUEL or QBE. These languages do not

require the user to know about structures in the physical organization of the data such as hashing functions, B-trees, inverted lists, etc. [Date 1987]. On the other hand, they provide *physical data independence* by translating a logical query into a physical query (i.e., a sequence of operations at the physical level). However, SQL, QUEL or QBE *force* a user to navigate the logical structure of the data (relations, records and sets, or entities and relationships). The user must know in which relations the attributes are stored and which attributes to join in order to navigate between relations. Most of these query languages do not provide *logical data independence.*

In our approach, we address the problems described above by developing a scenario in which analogy among databases plays a significant role during database definition and query formulation phases. Hence, we assume that the user is familiar with the objects (or entities) and the relationships in the application domain known as the *target domain.* She has at her disposal a manual that contains examples of database designs with a variety of database queries in some exemplary milieu called the *base domain.* The manual explains the example queries in simple terms and gives their implementation in a database query language such as SQL. Our user is not a database expert. She can neither define the database nor can she compose query implementations from scratch in any domain, target or base. At best she can construct simple queries in the base domain by copying them verbatim from the manual.

The cognitive processes of such a user who wants to define her database and then to derive the implementation of a target domain query specification may be described as follows:

Defining the database (Steps 1 until 4) :

1). The user maps the entities and relationships in the target application domain onto their conceptual counterparts in the base domain. The mapping is referred to as the *analogical observation* between the base and the target domains.

2). The user searches in the base domain the neighborhood of each object implied by the analogy developed in step 1). The neighborhood of an object, called also the *context of analogy*, is defined as the set of objects and relationships connected directly[1] to the objects involved in the analogical observation.

3). She then **selects** from neighborhood objects and relationships which will be mapped back into the target domain.

4). Finally, the user maps the relations implied by the analogy developed in steps 1) and 3) back from the base to the target domain, possibly making some adaptations in their expression there.

Reusing queries existing in the base domain (Steps 5 until 7)

5). By using the analogies developed during the definition of the target database, the user translates her query specification from the target to the base domain according to the mapping developed in steps 1) until 4). She finds in the manual descriptions of

[1] We consider two types of connections (by database relations and by semantic constructs; as discussed in sec. 2.5)

base queries that wholly or partially match the analogical image of her target query in the base domain. The manual provides implementations of each base query.

6). The user selects the base query which *best matches* the target query's image. She edits its implementation as necessary, relying on the manual for guidance, to make it correspond to the analogical image of the target query specification.

7). Finally the user maps the implementation back from the base to the target domain, possibly making some readjustment in its expression there. These readjustments may involve additional navigation in the database.

1.2 General Approach

Several practical arguments favor the strategy outlined above. Certain domains are widely used by the database community (e.g. [McFadden and Hoffer 1988]). As a result one can find a large body of exemplary databases with a set of queries and their associated implementations for such domains. Furthermore, real world domain experts are often quite good at identifying analogical mappings between domains. This means that a user will usually be able to perform step 1) without difficulty.

In order to define the target database (i.e. step 2) until step 4)), we used a variety of the structure-mapping approach developed by [Gentner 1983]. Rather than considering analogy as a mapping of causal networks, we propose to consider it as a mapping of objects, relationships and semantic constructs (e.g generalization, classification, etc.). In our approach we have assumed that the user is exposed to a complete knowledge concerning the

base domain. This knowledge is embedded into the base domain's conceptual model. A *semantic* data model is used to represent the base domain.

To perform query reuse (i.e. step 5) until step 7)) we used an alternative to more traditional, "deep" planning techniques which would treat the query as a planner's goal, and use planning techniques to derive a query implementation from scratch. The problem with full-scale planning, however, is that a planner has to be provided with knowledge about the activity of converting high-level specifications into low-level implementations (planning operators), and metarules for selection of alternative subplans emerging during planning (planner's control knowledge). Such knowledge is difficult to obtain and codify, and consequently a planner that would do the job is difficult to develop for a realistic domain. We propose a simplified technique, in which planning is only applied in a "shallow" manner, to derive simple SQL code from well defined and non-ambiguous parts of queries.

The technique used is in fact a weak[1] variety of the derivational analogy approach to plan reuse [Veloso and Carbonnell 1991]. A classical derivational planner is given a goal and operates step by step to generate a plan that satisfies the goal. At each step it attempts to meet the goal by analogizing some existing plan to the goal domain. Only when no appropriate 'canned' plans can be found does the planner proceed to derive one step of the plan. This derivation produces new subplans to be built. Derivational analogy is recursively applied to each subplan obtained, etc. SAFIR assumes that only very simple, elementary plans can be derived by the planner. The planner performs more advanced planning by reusing existing plan derivations. Although this approach limits the scope of problems that

[1] Weak because only small derivations can be generated by the planner.

can be solved, it is adequate to model the naive user as assumed. The approach used in this thesis is similar to case based reasoning techniques. Indeed, in our approach the case could be the plan with the logical and SQL forms of a given base query.

1.3 Problem Definition

The problem addressed by this thesis is described as follows :

> "SAFIR accepts two types of inputs. The first input consists in a brief description of the objects and relationships existing in the target domain. The system then defines the target database interactively with the user. On the other hand, the second type of input that SAFIR can accept is a target domain's query specification. The system then outputs its correspondent query implementation written in SQL."

As illustrated in figure 1, SAFIR is composed of two main components :

- *Database definition by analogy* module
- *Query reuse by analogy* module

The database definition module allows the definition of target databases after giving as input description of the objects and relationships existing in the target domain. The system interactively and incrementally infers and stores analogies (i.e. object correspondences base-target) which have been validated by the user. On the other hand, when a target query

specification is given to SAFIR, the Query Reuse module generates its correspondent query implementation based both on analogies developed during the definition of target database and on a set of queries existing in the base domain.

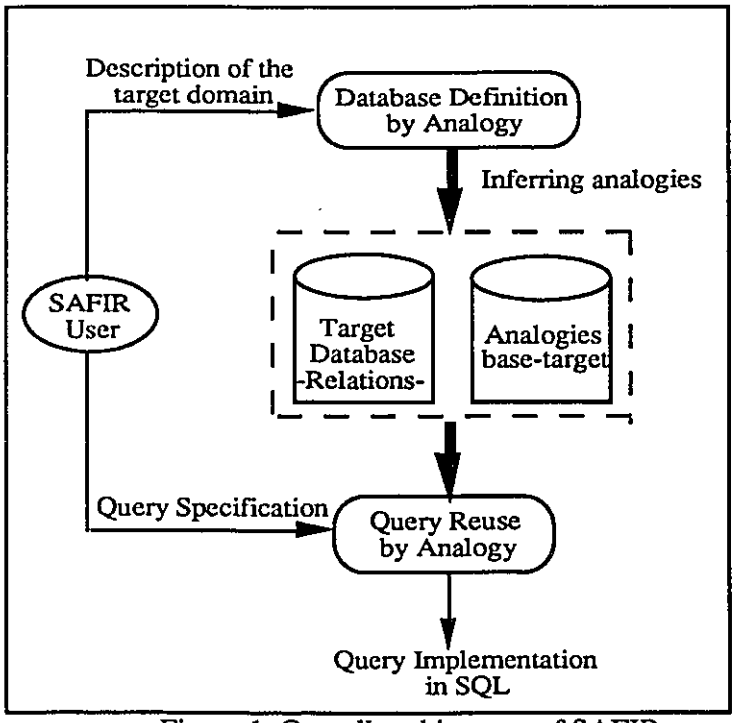

Figure 1. Overall architecture of SAFIR

Table 1 summarizes the knowledge used by SAFIR in order to perform database definition and query reuse.

Table 1. Knowledge used in SAFIR

Base Domain	Target Domain
• Semantic data model • Existing relations in the database • set of query specifications associated with their correspondent query implementations (in SQL) • Derivation history of a query implementation from its correspondent query specification	• Description of objects and relationships of the target domain. • Set of query specifications expressed in natural language.

1.4 Contribution of the thesis

This thesis explored and contributed in two main research topics :

- *Databases and Artificial Intelligence*
- *Software Reuse and Analogical Reasoning*

Current research on databases and artificial intelligence has investigated new applications of AI tools applied to database technology. Consequently, new research topics have emerged such as logic database, inferential DBMS, expert DBMS, deductive DBMS, knowledge base management system and so on [Desai 1990], [Date 1990]. However, little attention has been given to the role that analogy can play in databases. This thesis has explored particular

issues of analogical learning in the context of databases. Among these issues, we distinguish:

- *Database Design by analogy:*
- *Query Reuse by analogy*

On the other hand, a significant new trend is emerging within software reuse community over the past few years. That trend is toward applying analogical reasoning for reusing software artifacts [Miriyala and Harandi 1991], [Maiden and Sutcliffe 1991]. The work presented here contributed in this new research topic from the following points :

- Our work is significantly different from the few existing systems that use analogy. We tackle a different problem, the derivation of database relations and database queries from natural language questions, as opposed to the development of operating system macros or the design of information systems.
- An important difference with related work in software reuse is that the difference existing between a base and a target query is handled in the base domain.
- Another difference with existing methods in analogical reasoning research is the fact that we consider during the definition of the target database that analogy is a mapping of semantic constructs rather than a mapping of causality networks as used in many applications [Gentner 1983].

Chapter 2

Database Design by Analogy

This chapter describes the approach used for defining the target database by analogy to existing databases representing the base domains. First of all, we position SAFIR user comparatively to database user categories and then we describe in more details the analogical reasoner.

2.1 SAFIR User

Users of a database management system (DBMS) have been classified in different groups, depending on their knowledge and experience in using a DBMS. One of the most popular classification widely accepted by the database community is the three broad classes given by [Date 1990] and described as follows :

- Application programmer
- End user
- Database administrator

Recently, more works are conducted in order to improve the interface between the end user and the DBMS. Indeed, [Desai 1990] considers that the end users fall into two separate groups, naive users and on-line users. Naive users are users who need not to be aware of the presence of the database system. On-line users are persons who directly interact with the database via a terminal. On the other hand, a database administrator is a person who is responsible for creating, modifying, and maintaining the database.

From the scenario described in Chapter 1, we define our user called *SAFIR user* (see figure 2) as a person who combines both knowledge of the real world domain application and knowledge of the conceptual modeling techniques. As pointed out in the introduction chapter, our user is not a database expert, this means that neither can she be considered a database administrator nor is she a 'naive user' in terms of [Desai 1990].

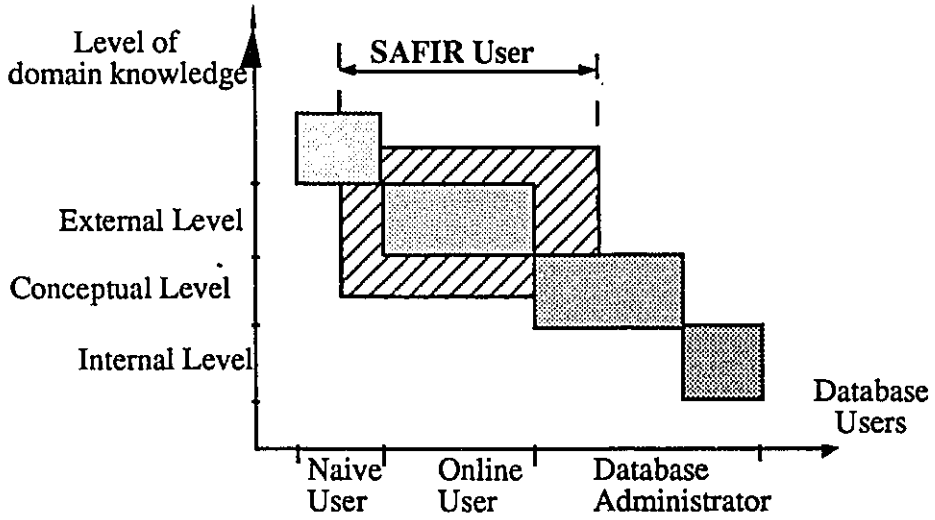

Figure 2. SAFIR user and database users

2.2 Overview of the Approach

The goal is to build (i.e. to define) interactively the target database by analogy to an existing database representing the base domain. The principle of the method, as shown in figure 3 is based on the analogical observation which represents the initial analogy given to the system. The process of acquiring analogies is then interactive and incremental. To obtain the existing relations in the target domain, the system builds a partial conceptual model of the target domain, called *virtual partial conceptual model*. --"Virtual" because the conceptual model in the target has not been derived from traditional and formal database conceptual modeling techniques, and "partial" because it reflects only some portions of the target domain.

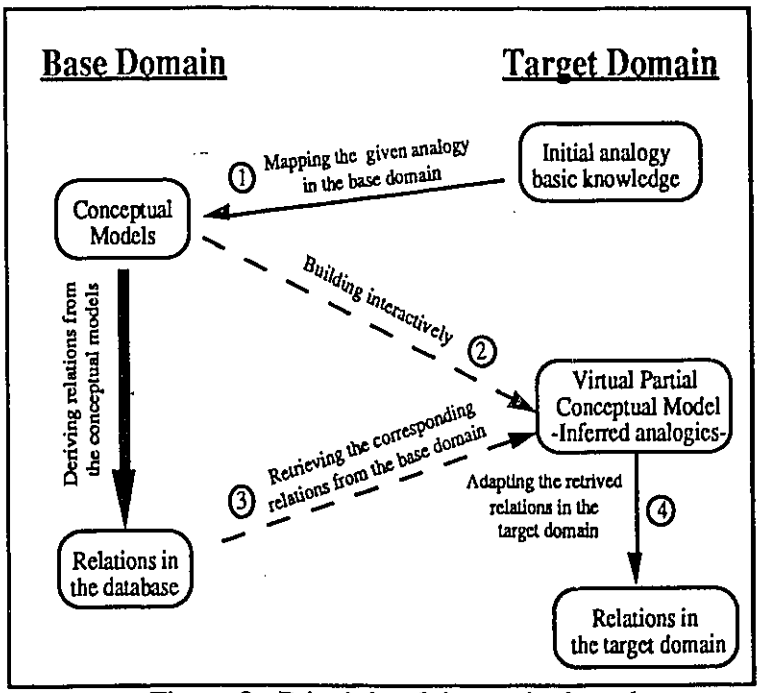

Figure 3. Principle of the method used
to derive relational schemes for the target database

The virtual partial conceptual model built in the target domain allows the retrieving of a set of relations existing in the base domain which have been implied by the analogy. These relations are then mapped back into the target domain by making some adaptations according to the objects and relationships existing in the target. As shown in figure 3, the process of drawing new analogies works in three stages. The first stage consists in acquiring the initial analogy (i.e. the analogical observation). The system identifies pairs of objects base-target involved in the initial analogy (i.e. step (1) shown in figure 3)). The second stage (i.e. step (2)) is to build the virtual partial conceptual model of the target domain by mapping parts of the base domain's conceptual model into the target domain. The next and final stage (i.e. steps (3) and (4)) is to derive the analogies at the implementation level (i.e. analogies involving relations between base and target databases).

2.2 Database Design Using a Semantic Data Model

In order to infer more analogies at the conceptual level, we represent the base and the target domains using a semantic data model. Indeed, semantic data models have some advantages that are not provided by other existing conceptual models such as network and hierarchical models [Peckham and Maryanski 1988], [Hull and King 1987]. Among these advantages we distinguish :

- Relationship representation
- Standard abstractions
- Network of hierarchies of relationships

- Derivation/Inheritance
- Degree of expression of relationship semantics
- Dynamic modeling

Many semantic data models provide direct representation of object types which are used to specify a number of atomic types. We distinguish two kinds of atomic types, abstract and printable types [Hull and Maryanski 1987]. Abstract types are typically used to specify physical objects in the real world, such as *Person*, or to specify conceptual objects such as *Business*. Printable (or representable) types are alphanumeric strings which represent subclasses of abstract types. *Person-Name* is an example of a printable type [Hull and Maryanski 1987].

Semantic data model provides type constructors for the creation of abstract objects such as generalization, aggregation, classification, and association. Generalization is the means by which differences among similar objects are ignored to form a higher order type in which the similarities can be emphasized. *Aggregation* is the means by which relationships between low-level types can be considered a higher level type. *Classification* is a form of abstraction in which a collection of objects is considered a higher level object class ('is-instance-of' relationship), and finally the *association* type constructor is a form of abstraction in which a relationship between member objects is considered a higher level set object (i.e. it expresses a relationship of type 'is-member-of').

2.2.1 Converting a Semantic Data Model into Logic

In our approach, we describe abstract and printable objects by using first-order logic. Object types are represented by predicates. Semantic constructs are described in logic as shown in table 2[1].

Table 2. Translation table from a semantic conceptual model to logic expressions

Semantic constructs	Equivalent logical description
Generalization	GEN(object1, Object2, ..,Objectn, Object)
Aggregation	AGG(object1, Object2, ..,Objectn, Object)
Classification	logical implication
Association	Prolog built-in predicate SETOF

As an example, the semantic data model represented in figure 4. is converted into logic as follows :

```
;; Generalization
gen(secretary_service, research_teaching_service, department)
;; Associations
setof[X, department(X) ∧ law_department(X), Z]
setof[X, department(X) ∧ engineering_department(X), Z]
;; Classification
secretary_service(X) ==> administration_service(X)
;;Attributes
professor(X) ∧ secretary(Y) ==> works-with(X,Y)
secretary(X) ∧ secretary_service(Y) ==> works-in(X,Y)
```

[1] Abstract types are depicted with triangles, atomic printable types are depicted with flattened ovals, and subtypes are depicted with circles

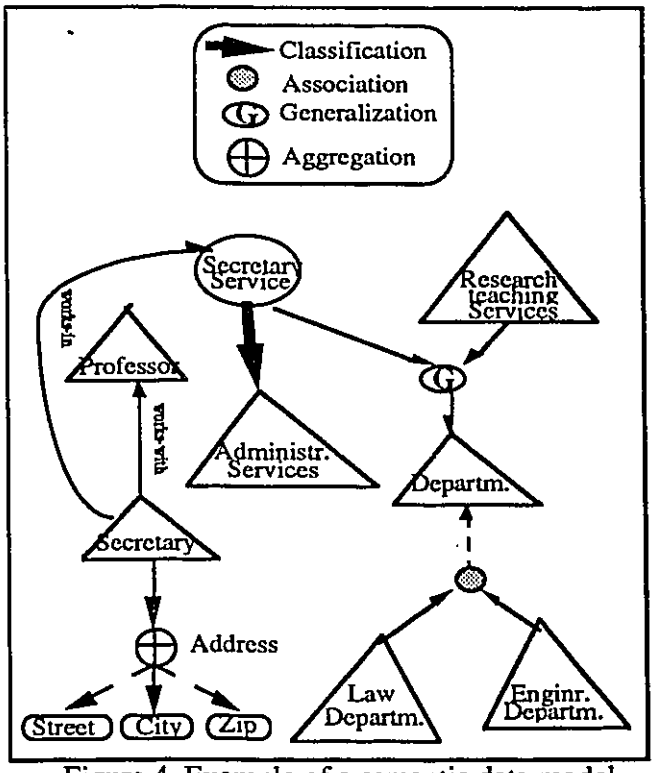

Figure 4. Example of a semantic data model

2.2.2 The Implementation level : Relational Database

At the implementation level, both the base and the target databases are represented using a relational data model. The relational data model is based on the mathematical concept of *relations*. It consists on three basic components [Date 1990, Desai 1990]:

- a set of domains and a set of relations
- operations (i.e. queries) on relations

Each relation is described using a set of attributes. For a given application, an attribute may only be allowed to take a value from a set of permissible values. This set of allowable values for the attribute is the domain of the attribute. A relation has two parts : a relation

scheme (or header), and a time-varying set of tuple (or body). A *tuple* represents an instance of the relation.

Generally, a subset of attributes of a given relation is used to distinguish between tuples of the same relation. This set of attributes is called the *key* of the relation. There may be more than one key in a relation; all such keys are known as *candidate keys*. One of the candidate keys is chosen and is called the *primary key*; the others are known as alternate keys.

Example

In the example given in section 2.2.1, the entity Professor is represented by a relation PROFESSOR(Prof#, Name, Phone-Number, Address), where Prof#, Name, Phone-Number, and address are the attributes of the relation PROFESSOR, and Prof# (Professor number) is the primary key.

2.3 Analogical Observation

Given a target domain, a naive user with little experience in defining databases, gives an initial analogy to the system called the *analogical observation.* We define an analogical observation as follows : "A naive user identifies particular analogical events in the **representation** of one or two base domains. These events may be represented in the database as entities or relationships or any semantic constructs (e.g classification, generalization etc.). The system generates a set of base-target pairs , which constitutes the

initial analogy resulting from the observation. By giving this initial analogy, the analogical reasoner identifies the existing relations in the database involved in the analogy."

Example

Let the base domain represent the Mountain View database which is a medical related database. The target domain is represented by the education database. The user identifies potential analogous events such as 'Professors teach courses' and 'Doctors treat patients'. These two events produce a set of pairs base-target summarized in table 3[1]:

Base Domain	Target Domain
(Patient,PATIENT)	(Course, COURSE)
(Doctor, DOCTOR)	(Professor,PROFESSOR)
(Treat, TREAT)	(Teach,TEACH)

Table 3. Pairs base-target and their relations existing in the two databases

In general, the user will focus only on some portions of the base and the target domains. This leads us to fragment the conceptual model of the two domains in different parts. The analogical observation represents then the analogy between some parts of the base domain and some parts of the target domain. Therefore during the elaboration of the analogy, the system maps portions of the base domain's conceptual model into the target.

2.4 Context of the Analogy

Central to the analogical reasoner is the context of analogy defined in the base domain. The context of analogy consists of a portion of the base domain's conceptual model, and it represents the set of objects connected directly to the objects involved in the analogical observation. As an example, in figure 5, the object Person is considered an object of the context of analogy because Person is connected directly to the object Patient involved in the analogical observation.

The context of analogy plays in fact the same role as plays the systematicity principle in the structure-mapping approach developed by [Gentner 1983]. Systematicity principle suggests that people prefer to map connected systems of relations rather than isolated objects. However, an important difference of our approach with Gentner's approach is the fact that in the structure-mapping method, the base domain is organized into a causal network. In our case, the base domain is represented by a conceptual model.

2.5 Virtual Partial Conceptual Model

The virtual partial conceptual model is built in the target domain. It results from the mapping of portions derived from the base domain's conceptual model. These portions represent in fact the mapping of one (or more than one) context of analogy. All the objects of the partial virtual conceptual model have their analogous counterparts in the base domain. Moreover, some objects in the base are mapped into themselves in the target domain. In this case, these objects are also objects of the target domain.

Example :

The example shown in figure 5 describes a virtual partial conceptual model built in the target domain. For example, the object Person, defined in the base domain, is mapped into the target object Person (i.e. the object Person is shared between the base and the target domains). On the other hand, the object 'Research assistant' is built in the target by analogy to the object 'Nurse' existing in the context of analogy which has been defined in the base domain.

As mentioned in section 2.2, figure 5 describes two kinds of object correspondences base-target obtained either from the initial analogy such as 'Professors teach courses' is analogous to 'Doctors treat Patients' (i.e. step 1 shown in figure 3, see sec. 2.2), or from the mapping of the context of analogy into the target domain (i.e. step 2 shown in figure 3, see sec. 2.2). However, not all objects existing in the context of analogy have their analogous counterparts in the target domain. The user can accept or reject some inferred analogies even if semantically they are accepted in the target domain. As an example, the concept of 'Speciality' and 'Staff' are not considered as objects of the target domain because the user doesn't need in her application such description for the objects Professor and Employee.

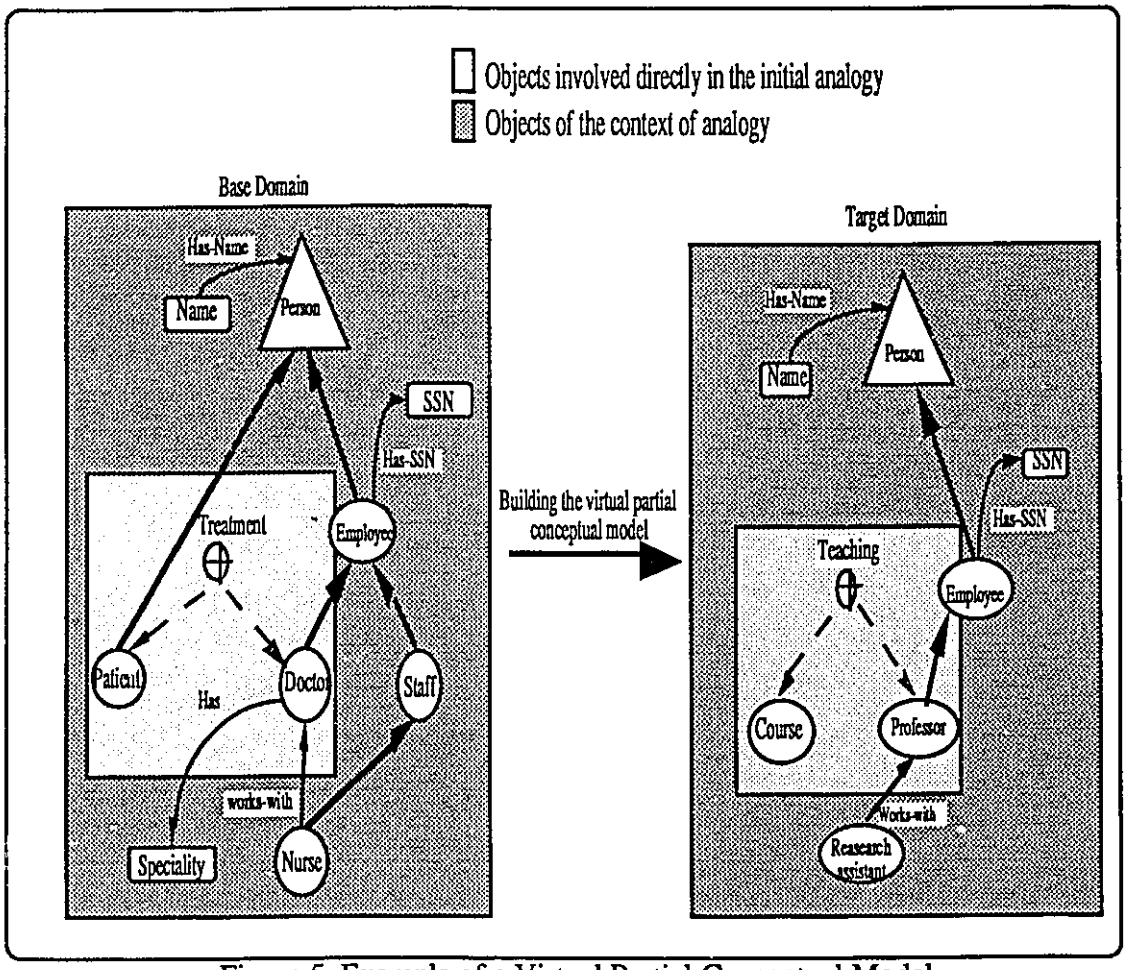

Figure 5. Example of a Virtual Partial Conceptual Model

2.6 Analogical Reasoning

Reasoning by analogy is the process of building the set of candidate inferences which are logic expressions mapped, by analogy, from the base domain into other logic expressions defined in the target domain. In this section we describe in more detail the process of drawing analogies among databases.

2.6.1 Recognition of Analogy

Recognition addresses the problem of identifying the potential analogous candidates (in the base domain) by giving basic description of a target domain. By allowing partial similarity between target and candidate bases, a central problem of recognition is to impose constraints on the retrieval of base analogs. Recognition proceeds by extracting portions of the base domain's conceptual model, represented as a set of logic expressions which will be mapped in the target. Initially, recognition is done through the analogical observation given by the user. Therefore, the system infers a set of analogies (base-target) at the implementation level. Acquiring the analogical observation is summarized in stage one given below:

Stage 1 : Acquiring the initial analogical observation

1. Acquire basic description of the objects existing in the target domain
2. The user selects a base domain and the system displays basic relationships retrieved from the base domain.
3. The user selects a relationship in the base domain and gives its analogous relationship in the target domain.
4. The system builds a set of object base-target and displays the current status of inferred analogies.

2.6.2 Elaboration of Analogy

Elaboration addresses a general problem: finding an analogical mapping between elements of target and base domains. Elaboration incrementally extends a mapping between target and base domains that supports transfer of analogical inferences, and the effectiveness of these extensions is subject to evaluation.

Elaboration starts with a partial mapping uncovered during recognition. Priority is given to the objects appearing in the context of analogy. The elaboration of the analogy is achieved in two interrelated stages. First, the elaboration process works at the conceptual level. The next stage is the elaboration of the analogy at the implementation level.

2.6.2.1 Elaboration at the Conceptual Level

Elaboration of the analogy at the conceptual level starts by acquiring information on the properties of the objects implied by the analogical observation. Once these properties have been captured, the system starts by mapping the context of analogy in the target domain. The result is a partial virtual conceptual model of the target domain. At this stage, the user can confirm, reject or modify the analogies built from the mapping. Hence, the mapping of the context of analogy is in fact a mapping of semantic constructs used in representing the base domain (e.g. generalization). Furthermore, by modifying the context of analogy in the target, the user is implicitly building portions of the target domain's conceptual model. Another important characteristic of the mapping is the fact that the system is incrementally acquiring analogies (base-target) which involve objects existing at the conceptual model.

On the other hand, once the context of analogy is modified and therefore the analogies are confirmed by the user, SAFIR extends the mapping on the objects connected directly to the objects existing in the context of the analogy. We remark here that initially the system starts the mapping based only on the objects involved in the analogical observation and then it extends the mapping on the objects existing in the context of analogy. When reaching this level, the objects connected directly to the objects of the context of analogy are then considered in the mapping base-target. To mark the end of the elaboration at the conceptual level stage, the user stops explicitly the process of deriving analogies between the base and the target domains.

Following is a summary of the different steps describing the elaboration of analogy at the conceptual model :

Stage 2 : Elaboration of candidate inferences at the conceptual level

1. Sets the initial context of analogy (derived from stage 1), and displays it.
2. The user confirms, rejects or modifies the context of analogy once mapped into the target domain.
3. Once the user adds descriptions of new objects in the target (it may happen that these objects have been already described), look for their equivalent analogous objects in the base. The search starts by matching the logic expressions of the added information against objects existing in the selected base domain (see step 2 of stage 1).

The mapping is constrained in order to restrict the enormous space of possible mappings between base and target to a smaller space of plausible or useful mappings. In many approaches (e.g. [Hall 1989, Maiden and Sutcliffe 1991]) constraining addresses the problem of *'what to preserve in the analogical mapping?'*. In our approach, the mapping is constrained semantically, structurally, and contextually. Semantic constraints address the problem of what semantic structures (of base and target knowledge) are commonly preserved during the analogical mapping [Maiden and Sutcliffe 1991]. In our case, we tend to preserve semantic constructs such as aggregation, generalization and classification. Structural constraints tend to preserve the relational structure of the source representation (e.g. [Gentner 1983, 1987]). Contextual relevance constraints focus on the contextual relevance of mapped elements asking which relational or semantic structures to preserve within the current reasoning context. In our case, the contextual relevance constraints depend on the current context of analogy and also on the fact that the user can decide to change the direction taken by the analogical reasoner. Indeed, the user can explore new analogies between base and target by introducing new description of the target or by ending explicitly the elaboration of analogies process (i.e. ending the elaboration process means that the user wants to explore analogies at the implementation level). In general the contextual relevance constraints focus on those facts which are important at the moment (e.g. [Hobbs 1983a, 1983b]).

2.6.6.2 Elaboration at the Implementation Level

Once the elaboration of analogy at the conceptual level is ended by the user, the system starts the elaboration at the implementation level. This step consists in building the set of

relations making up the target database. The process starts first by retrieving the base relation implied by the analogies developed during the elaboration process (i.e. elaboration at the conceptual level). These relations are then mapped back into the target domain by making some readjustment in their expression depending on the objects and relationships existing in the target and also depending on some relational database constraints such as the key of the relation must functionally determine all the attributes of the relation.

2.6.3 Evaluation of the Analogy

Evaluation of the analogical inferences is achieved by asking the following questions :

" How to confirm inferences extended from base to target ?"

" How to repair inappropriate extensions ?"

The candidate inferences must be evaluated in order to be accepted or rejected. In our approach, the evaluation is performed by the user. She is the only person who can decide if she will maintain the relations and the analogies derived during the analogical reasoning process. However, the rejected analogies are no longer used during the analogical reasoning process.

In the literature, analogical inferences are confirmed using various test predictions about the target domain. Some of these methods employ a systematic approach to justify analogical mappings. As an example, [Winston 1980] uses abductive reasoning to verify an analogical inference. More ambitious evaluative strategies weigh the *problem solving utility* of

analogical inferences. Indeed, Carbonell's transformational analogy mechanism uses a similarity metric to select operators (called T-operators) which incrementally transform a source solution sequence into a target solution sequence. [Greiner 1985] uses a different approach within a deductive problem solving framework. In general, justification gives representational descriptions of the reasons which support an inference or action in some domain [Hall 1989].

2.7 Detailed Example

The following example is a (partial) session in which the user is interactively defining the target domain objects and the system is incrementally inferring new analogies depending on the current context of the analogy.

Let the base domain be the Mountain View database [McFadden and Hoffer 1988], and the target domain be the Education database [Ross 1988]. Semantic data model for the base domain appears in Figure 6.

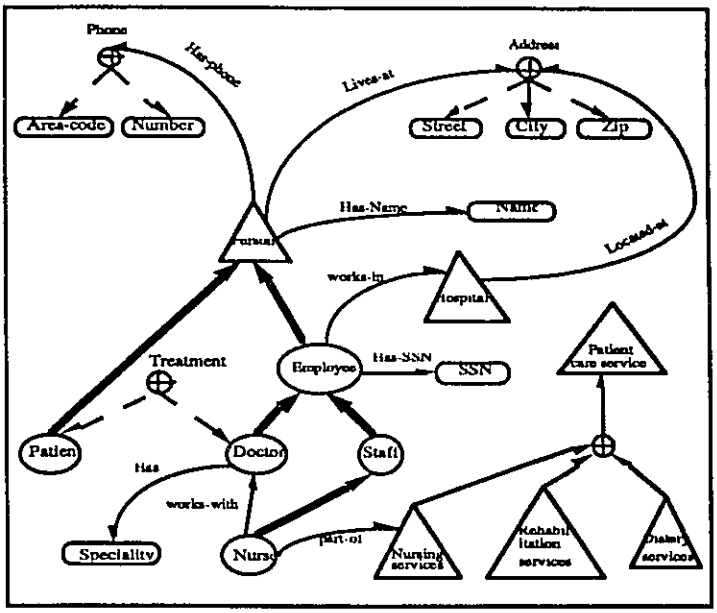

Figure 6. The base domain , Mountain View database

Logical representation of the base domain

agg[street, city, zip, address]
agg[area-code, number, phone]
agg[patient, doctor, treatment]
gen[nursing_services,rehabilitation_services,diatery_services, patient_care_services]
nurse(X) ==> stuff(X) ;
doctor(X) v staff(X) => employee(X)
patient(P) v employee(E) ==> person(X)
employee(E) ∧ hospital(H) ==> works-in(E,H)
person(X) ∧ address(A) ==> lives-at(X,A)
person(X) ∧ phone(P) ==> has-phone(X,P)
nurse(N) ∧ nursing_services(S) ==> assigned(N,S)
nurse(N) ∧ doctor(D) ==> works-with(N,D)
hospital(H) ∧ address(A) ==> located-at(H,A)

In the following, "U > " stands for the user and "S> " stands for the system. Comments are italicized.

Stage 1 : Acquiring the Analogical Observation

```
S> Please define possible objects in the target domain :
U> professor, student, researcher, department.
S> Which object do you want to describe ?
U> professor.
S> Describe relationship(s) between professor and other objects in the
target  domain ?
U> professors teach courses.
S> course is an unknown object. Do you want it to be considered an object of
the target domain ?
U> yes.
S> Do you want to see relationships in the source ?
U> yes.
S> Please select the Base domain to be displayed :

        1. Mountain view database
        2. Library database
U> 1.
S> In the Mountain View database I know the following relationships :

        1. doctors treat patients
        2. nurses work with doctors
        3. employees work in hospital
        4. persons live at an address
        5. hospital are located at an address

S> List other relationships (yes or no) ?
U> no
S> Describe a relationship from the base domain analogous to:

                        'professors teach courses ?'

U> doctors treat patients ?
S> considering the analogy :


             Base   (doctors  treat patients) -->
             Target (professors teach courses)


    Constructing pair correspondences between source --> target :

    doctor  --> professor ; patient --> course ;  treat   --> teach
```

Stage 2 : Elaboration at the Conceptual Level

Context of the Analogy:

```
1. GEN(doctor, Staff, employee),
   "Doctor and staff are employees"
2. GEN(patient, employee, person),
   "patient and employee are persons"
3. nurse(X) ∧ doctor(Y) ==> works-with(X,Y),
   "nurses work-with doctors"
4. AGG(doctor, patient, treatment),
   "doctors treat patients"
5. PROPERTY(doctor,speciality)
```

Display the current status of inferred analogies :

Analogical Inferences:

```
AGG[doctor, patient, treatment]  -->
AGG[professor, course, teaching]
```

The system maps the context of analogy in the target domain, capitalized letters refer to the objects from the base mapped into themselves.

Base	----->	Target
1. doctor is an employee		1. professor is "EMPLOYEE"
2. patients and employee are persons		2. courses and "EMPLOYEES" are "PERSONS"
3. nurse works with doctors		3. "NURSE" "WORK-WITH" professor
4. Doctor has a specialty		4. Professor has a "SPECIALTY".

We remark that the expression 3) is mapped into the target by keeping the relationship "WORK-WITH" which can be accepted or changed by the user. As an example, the user can specify that "NURSE" "works-with" professor has to be changed to "RESEARCH ASSISTANT" "WORK-WITH" professor.

```
S> Which statements do you want to consider :
U> 1
S>  What could be "EMPLOYEE" in the target ?
```

U> employee
S> employee in now considered as an object of the target domain.

Analogical inferences:

AGG(doctor, patient, treatment) -->
AGG(professor, course, teaching)
GEN(doctor, Staff, employee) -->
GEN(professor, employee)

Context of the analogy:

1. GEN(doctor, Staff, employee),
   "Doctor and staff are employees"
2. GEN(patient, employee, person),
   "patient and employee are persons"
3. nurse(X) ∧ doctor(Y) ==> works-with(X,Y),
   " nurses work-with doctors"
4. hospital(H) ∧ employee(X) ==> works-in(X,Y)
   "employee works in a hospital"

...

Here the user ends the incremental building of analogies. The next step is inferring the correspondent analogies at the implementation level.

Stage 2 : Elaboration at the Implementation Level

The existing relations in the source databases implied by the analogical inferences :

Mountain view Database

relation_Doctor(SSN,Name,Speciality)
relation_Patient(Pat#, Name, Address, Birthyear, SSN)
relation_employee(Emp#, name, Job-code, salary)
treatment(SSN, PAT#, Date)

Library Database

relation_book(ISBN, author, title, sales, publisher, year-publication)
relation_journal(J#, name, volume, serie, year)
relation_conference(C#, title, sponsor, place)

Creating the relations: professor, course, teaching :

relation_professor(P#, name, address, speciality)

```
      P# has been created analogically to SSN and ISBN.

Analogical inferences (implementation level):

      relation_doctor --> relation_professor ;
      SSN --> P#, Name(doctor) --> Name(professor),
      speciality(doctor) --> Speciality(professor)

      relation_book --> relation_professor
      ISBN --> P#,
      relation_journal --> relation_research_assistant
      Student-id# --> P#
      relation_conference --> relation_research_assistant
      Emp# --> P#
      publication --> researcher
```

At this stage the user confirms the relations built in the target.

Chapter 3

Query Reuse by Analogy

This chapter presents the second important part of the thesis concerning applying analogy for database query reuse. We show how queries initially developed for the base domain are effectively reused to answer target domain' query's specifications. The reuse is done by taking advantage of the analogies and the relations developed during the definition process of the target database described in the previous chapter.

3.1 Overview of the Approach

SAFIR allows the reuse of query implementations (written in SQL) defined in the base domain. The system is given as input a target query specification and as output its correspondent implementation (in SQL). We rely on a weak form of derivational analogy -- traces of a planner operating in the base domain are used to derive a target domain' query's implementation in SQL from its specification. The very simple planner used builds in the base domain plans that result in particularly simple, prototypical queries. Our approach

consists of two steps. In the first step, the target specification is translated into the base domain, and the closest base specification is retrieved, together with its associated query implementation. In the second step, the base query implementation is modified to accommodate mis-matching parts of the target. The resulting modified base query implementation is then transferred to the target domain.

Figure 7 below represents different layers of representation used in SAFIR for query reuse. Top level query specifications are expressed in natural language. These specifications are translated into a logical representation (the logic layer in Figure 7) using XCHAT-80: a version of CHAT-80 extended for SAFIR. CHAT-80 system is a prototype natural language question answering system, developed by [Warren and Pereira 1982] for deriving a logic expression from a query expressed in natural language.

Implemented in PROLOG, CHAT-80 system translates English questions into the Prolog subset of logic. The resulting logical expression is then transformed by a planning algorithm into efficient prolog question which will be executed to yield the answer.

In the third layer down, a tuple-based plan representation is derived from the logical query expression. This tuple-oriented representation is used to derive the SQL form of the query specification [Date 1990, Desai 1990], whose implementation is represented on the lowest layer of figure 7.

The symbols on the left and right margins of figure 1 have the following meaning:

Q<Domain> : Query specification in a logical form. <Domain> indicates whether the domain is the base or the target domain.

S<Domain> : Query implementation written in a real query language such as SQL.

P<Domain> : Derivation of the query implementation

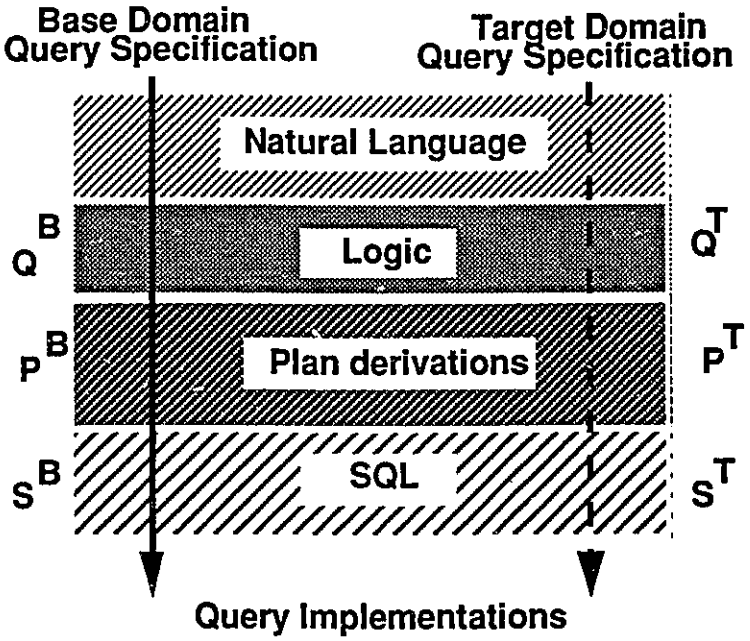

Figure 7. Illustration of the basic objects handled by SAFIR and their symbolic notation from the query specification; represented as a trace of the planner.

3.2 Definition of the Analogy

We base our concept of analogy on the classical schema presented in [Kodratoff 1990](see Figure 8). In the SAFIR context, the schema shows the mapping of a query specification from the target domain into the base domain (application of the inverse of the operator α). Once a query in the base domain is selected, the corresponding plan is retrieved using the

operator β. β takes a given Q^B and its corresponding P^B and obtains the implementation S^B contained in P^B. At this stage, the retrieved plan is transferred to the target by applying the operator α', which is similar to α but operates between the implementations, rather than the specifications .

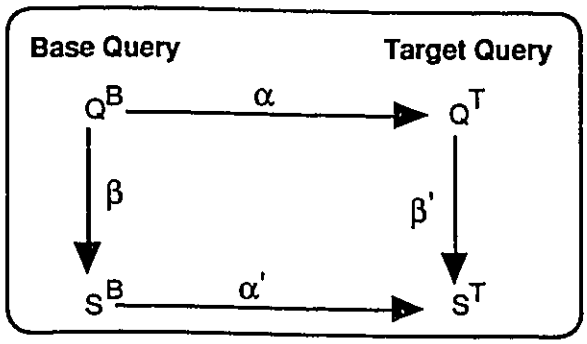

Figure 8. The analogical mappings used by SAFIR

The derivation function (denoted by δ) of a query implementation (S^T) from its query specification(Q^T) is described below :

$$\delta: Q^T \dashrightarrow S^T = \begin{cases} \alpha.\beta.\alpha^{-1} \text{ if } Q^T \in Rng(\alpha) \\ \text{derivational analogy otherwise} \end{cases}$$

When the operator α does not select a relevant base case (i.e. the base case does not represent completely the target case), we perform the derivational analogy process.

Derivational analogy (DA) was developed by [Carbonell 1986] to investigate analogical reasoning in the context of problem solving. The derivational analogy technique solves a new problem by making use of a solution derivation that was generated while solving a previous problem. The new problem is then solved by recreating sequences of decisions and justifications for decisions that were used to solve a precedent problem.

Carbonell uses derivations in a way similar to the manner in which Gentner uses causal networks. Carbonell proposes transferring derivations between examples whereas Gentner proposes transferring causal network.

As described in [Mostow 1987], derivational analogy aims to solve a problem by *replaying the plan* used to solve previous problem, modifying it where necessary. DA emphasizes the *reuse* of the process rather than the product of design. Most of the recent work on analogical problem solving is based on Carbonell's work on transformational and derivational analogy.

Transformational analogy mechanism illustrated by Carbonnell was initially implemented in a production system model of the subject's behavior [Carbonell 1983, 1986]. The idea is to generate proofs in geometry and simple number theory from proofs of related theorems. As an example, proving that Odd*Odd= Odd is achieved by patching and replaying the proof of Even*Even = Even.

Derivational analogy method was proposed to remedy the representation deficiencies of transformational analogy [Carbonell 1986, Mostow 1987]. It represents a problem solving plan as a hierarchical goal structure, showing how and why each goal was decomposed into subgoals. It solves a new problem by replaying the plan top-down. When the subplan for a subgoal fails, the plan is patched by solving that subgoal from scratch.

SAFIR's planner has only limited planning capabilities. It first identifies a query Q^T which is similar to Q^T in the Rng(α), and it will patch the differences between the Q^B and the $Q^{B'}$ that corresponds to $Q^{T'}$ through α in the base domain. According to the assumptions set out

in section 1.1, only very simple subplans can be developed to address these differences. Thus this derivation function can only be successfully applied to target queries which deviate from the existing base query by only one object[1] .

3.3 Selection of Base Case

The method used to select base cases is similar to that used in [Miriyala and Harandi 1991]. The selection of the case which corresponds to a target case Q^T is done from among cases in the base domain that are "close" to $\alpha^{-1}(Q^T)$. A base case Q^B is considered close to the counter image of Q^T under α when there is a partial match between $\alpha^{-1}(Q^T)$ and Q^B. Closeness is measured by associating with each such query a variable called the *mismatch* describing the number of objects by which the two cases differ. By varying the range of mismatch we can limit or extend the number of candidate base queries which approximate $\alpha^{-1}(Q^T)$. Unmatched objects constitute the difference between the base and target queries. Figure 9 describes three classes of differences between the base and the target cases which SAFIR distinguishes. In figure 9(a), the difference is contained entirely within the target. The difference is therefore added to the selected base case after the target query is mapped to the base domain. Addition is performed by modifying the plan corresponding to the base case. This consists of applying a set of operators to the intermediate forms of the base case stored in the plan. In figure 9(b) the difference is in the base case. The modification applied to the base case consists of deleting the difference from the base case. The third case described in figure 9(c) represents a situation where the base case or the target case are not a part of each other. In this situation, SAFIR first deletes the difference, appearing in the base

[1] An object consists in one predicate.

case selected, and it adds the difference found in the target to the selected plan corresponding to the base case.

In the case where exists more than one base query which minimizes the mismatch, SAFIR uses heuristics to select the best query. The "best"[1] is the easiest to modify. The ease of adaptation is measured by the effort required to implement the difference. In our implementation, the main heuristic used is to prefer the base queries which approximates the target query only by one object. Furthermore, when the difference is in the base, we prefer the base query which doesn't contain the operator Exist. Indeed, translating the operator 'Exist' from logic to SQL requires more deep planning techniques which are not supported by SAFIR's planner.

[1] This approach is similar to Krueger's cognitive distance concept [Krueger 92] where the objective is to spend less effort in deriving the implementation part related to the difference.

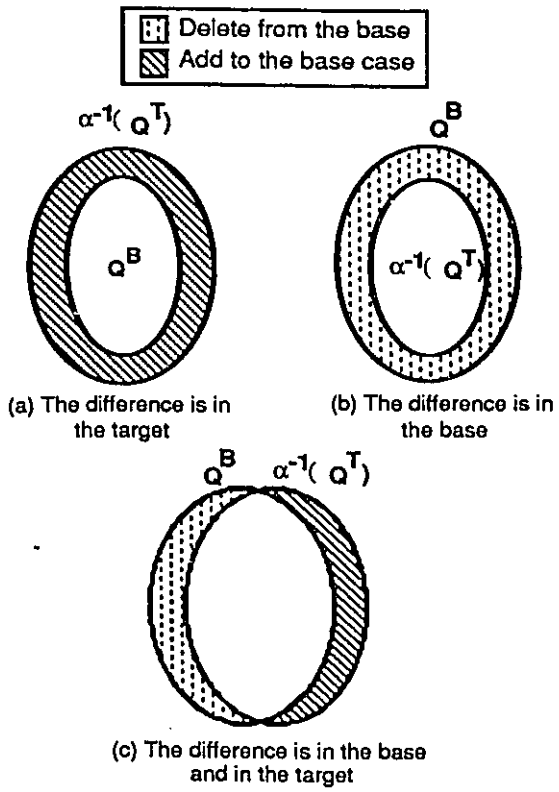

Figure 9. Difference between base and target query

3.4 Reuse Process

The reuse process is used to derive queries in the target domain from ones already existing in the base domain. The system has limited knowledge about deriving complex queries. The derivation history is represented by a plan which describes the answer to the query (referred to as the goal), and all the steps which perform the main transformations of the query from specification to implementation. A plan contains the following information:

—the answer clause, which represents all the necessary information to describe the answer to the query,

—the representation of the traces of the derivation process,

—the final SQL form of the query.

The structure of the plan appears below :

Plan Name of the plan corresponding to the query specification

:Query_Specification
The logical form of the query produced by XCHAT-80.

:Answer

:Variable_Implementation
Transformation sequence of the answer variables, from their logical description to their tuple-oriented representation.

:Derivation
The derivation process of the query implementation from the query specification

:Steps
This clause contains all the intermediate steps used in deriving the query implementation.

:Transformations
Contains the transformations which transform an object from its logical description to its database representation (relation or attribute)

:Constraint_Justification
Describes the justifications of the constraints used in the tuple expression of the query represented in the **Steps** clause.

:Difference
The difference between the base and target cases

:SQL_form
Contains the SQL form of the query specification.

3.4.1 Derivation Process

An XCHAT-80 query in the base domain will first be transformed into a tuple-oriented expression and then into its equivalent SQL form. We assume that the system can perform only basic derivation steps for simple prototype queries. A query specification is translated from the logical form produced by XCHAT-80 to a tuple expression by producing all the intermediary forms of the query. These forms are generated by analogical transformations given to the system or acquired by it. Thanks to these transformations it will be less time-consuming to handle small differences between the base and the target cases than it would be to derive the plan corresponding to the target query specification from scratch.

3.4.2 Planning Process

The goal in using a planning process is to derive a query implementation from a given query specification where possibly exists a difference between the base and the target query specifications. The planner proceeds by extracting information, called initial world from the plan corresponding to the base query specification which best matches the target query specification (see figure 10). This information represents the derivation steps of the query specification to its correspondent query implementation in SQL. The initial world is represented as a set of facts concerning the derivation history of the base query implementation. SAFIR's planner applies then a set of rules (Prolog rules) on the initial world to generate a sequence of actions to be applied both on the base query's derivation steps and on the existing the difference between the base and target query specifications.

The result is a query implementation in the base domain corresponding to base query specification taking into account the difference.

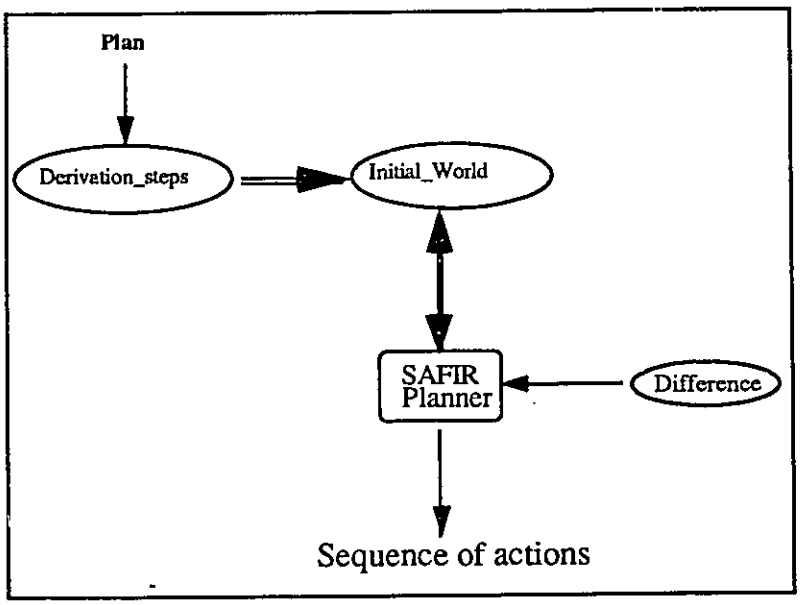

Figure 10. Principle of SAFIR's Planner

Using the notations described in section 3.2, the problem of deriving an implementation of a target query specification is formulated as follows :

> "Given a target query Q^T, a retrieved plan P^B corresponding to the base query Q^B and a difference DIFF, the problem is to find a sequence of actions SEQ such that applying SEQ on the difference DIFF and on the derivation steps (drawn from the plan P^B), we can derive an implementation in the base domain S'^B such that $S'^B=\alpha'^{-1}(S^T)$ where S^T is an implementation of the target query specification Q^T."

One of the main functions used by Safir's Planner is the 'Derive_Implementation' function which generates an implementation part (in SQL) correspondent to the difference. The following is a pseudo algorithm describing Derive_Implementation fuction :

```
Fuction Derive_Implementation_Diff(DIFF)

        /* This function generates an implementation part for the
           difference. It builds facts (called Difference_World) concerning
           each derivation step to be added to the plan. */

transform_diff=false;

/* transform_diff is a boolean variable which indicates whether it is necessary to
generate an SQL part for the difference */

/* previously_used(symbol) is a function which is true if symbol has been previously used
in the base query and false otherwise. Predicate_name is the name of the object
composing the difference */

if previously_used(predicate_name)

        /* predicate_name is the name of the object making up the difference */
then
        /* check if the variables refer to the objects previously defined */

        for i=0 to nv
        do
                if previously_used(var[i])
                then  /* it exists one or more than one variable which has
                         not been used in the previous objects */
                      transform_diff = true

                      /* recording the position of the object to be transformed later on */

                      transform_variable[j]=i;

                else  /* all the variables have been used. Therefore the derivation will
                      generate a constraint to be added in the WHERE clause of the SQL
                      query */

                      transform_diff = false
                endif;
```

```
        enddo
else transform_diff = true
endif

if not transform_diff
then   /* retrieve facts concerning the previous transformation of the bject
          composing the difference */

       for i=1 to nv
       do
              if constant_variable(var[i])
              then
                   /* add the constraint "attribute=constant" to the Derivation_World */

                      retrieve_attribute(predicate_name, var[i], Attribute);
                      add_constraint(Difference_World, DIFF, 'Attribute = var[i]');
              else
                      /* In this case a join is necessary with the previous object */

       retrieve_relation_name(predicate_name, 'current',Relation_name);
       retrieve_relation_name(predicate_name,'previous',Previous_relation);
       retrieve_attribute(predicate_name, var[i], Attribute);
       retrieve_joint_attribute(Relation_name,Previous_relation,Attribute_1,
                                Attribute_2);

       if Attribute_1 = var[i]
       then add_constarint(Difference_World, DIFF, 'Attribute_1=Attribute_2');
       else  generate_sql_part(DIFF, var[i], Attribute_2);
       endif
endif
enddo
else generate_sql_part(DIFF, transform_variable)
enddo

End function
```

The operator which comes with the difference could be of different types. For example, a difference operator could be the NOT operator. For each type of operator we apply a particular set of Prolog rules in order to derive an implementation part of the difference. These rules define a particular strategy for a given operator. The following are examples of

rules used to derive an implementation of the difference having respectively NOT and NUMBEROF as operators.

Action : Construct Internal_Constraint
(Object, Imp_var / Attribute = Var)

/* This rule expresses the condition that a given implementation variable must be equal to a generalized variable for the correspondent attributes. */

If Var is a constant_variable **and**
attribute_var(Var, Object, Attribute) **and**
transform_variable(Var, Imp_Var)
then
Add Internal_constraint(Imp_Var/Attribute=Var)

Action : Construct Numberof_SQL
(Numberof(Object), COUNT(Object_Implement)

/* This rule translates the logical operator numberof into the SQL COUNT operator*/

If Transform(Object, Relation. relation)**and**
attribute(Object, Relation, Attribute)
then
Add Numberof_SQL(COUNT(Relation.Attribute)

3.5 Examples

3.5.1 Example 1 : Education and Hospital databases

The following example highlights the basic concepts discussed in the previous sections. Let the base domain be the Mountain View database [McFadden and Hoffer 1988] and the target domain be the Education database [Ross 1988]. Analogies between these domains can be identified from the definition of the target database (see chapter 2). As an example the set of analogies stored by SAFIR may include the analogy that the entity Doctor plays the same role in the Mountain View database as the entity Professor does in the Education

database. According to this analogy Patient is considered the counterpart of Course. This creates an analogy at the implementation level between the relations involved, telling us that the relation PATIENT is analogous to the relation COURSE whenever the entities Patient and Course are involved in a query. Let us consider in our examples the following analogous object pairs given to the system by the external observer[1] .

Base Domain	Target Domain
(Patient,PATIENT)	(Course, COURSE)
(Doctor, DOCTOR)	(Professor,PROFESSOR)
(Treat, TREAT)	(Teach,TEACH)

The query below, stored together with its plan, logical form and SQL query implementation, is an example of a base case:

Query : What are the names of patients treated by doctor d1 ?

From this statement XCHAT-80 produces the logical form Q^{β}:

result(P,CC)<= setof(C,name(P,C) &Patient(P) & treat(d1,P),CC)

The answer to this query is given by the instantiation of the logical variables P and CC with values retrieved from the database. CC represents the name of a patient P. XCHAT-80

1 In sequel, capitalized names (e.g PATIENT) denote existing database relations, while normal names (e.g Patient) denote objects in the Entity-Relationships model.

translates the names of a patient P as a set of P, so the logical form of the query contains the Prolog predicate setof[2] .

Suppose that the user submits the following query in the *target domain* :

Target Domain Query:

What are the names of courses taught by p1 which are not taught by p2?

XCHAT-80 produces the corresponding logical form of the query Q^T in which the constants are generalized to new variables, V1' and V2' under the generalization: [V1'=p1 , V2'=p2].

result(P, NN) <= course(P) & teach(V1',P) & Setof(N-P,Name(P,N),NN) & Not(teach(V2',P))

By using the knowledge concerned the analogy between the base and the target domains, SAFIR applies the inverse of the operator α. The result will be a query in the base domain[1] :

$\alpha^{-1}(Q^T)$ = result(P, NN) <= Patient(P) & treat(V1,P) &Setof(N, Name(P,N),NN) & Not(treat(V2,P))

---

2 Setof is a Prolog built-in predicate, which has three arguments. The first argument is a template, represented by a term. The second argument is a goal and the third argument is a non-empty set (generally the third argument is a variable). Setof returns a set of all instances of template such that goal, given as a second argument, is provable.

1 In this example we suppose that the name of a Patient has a similar use with the name of a Professor.

The system searches the base library and selects a query that approximates our target query with one missing object. The difference is then detected (in the mapped target query, after performing α^{-1}). The query selected is Q^B. The difference clause in the plan associated with this query will contain the difference to be added to all the derivation steps described in the query's derivation clause. The retrieving and modification of the selected plan is performed by an application of the operator β.

Once a difference is found in the difference clause, SAFIR resolves it by performing some basic transformations from its repertoire of 'simple plans'. The result will be added to each step of the query derivation. The modified plan (base case) is then mapped to the target case without any modification. Therefore, the application of the operator α' produces the SQL form of the target query as shown in figure 11:

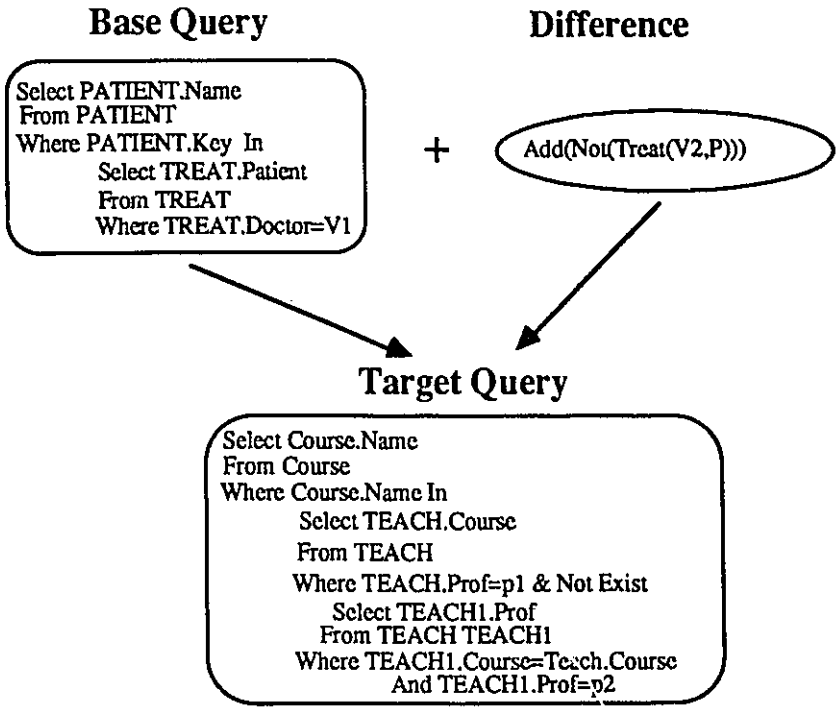

Figure 11. Example of a target query implementation derived from a base query and a difference.

3.5.2 Example 2: The Supplier and Education Databases

This example concerns the supplier-part-project database considered as the base domain, and the education database which is the target domain as described in the previous example.

The supplier database consists of four relations, the supplier, parts, projects and the relation shipment. The schema description of these relations is given below :

Relation_Supplier(Sup#, Supplier_Name)
Relation_Parts(Part#, Part_Name)
Relation_project(Prj#, Project_name)
Relation_Shipment(Prj#, Sup#, Part#),

Where Sup#, Part# and Prj# are respectively the primary keys of the relations Relation_Supplier, Relation_Part and Relation_Project.

Let us suppose that the initial analogy between the supplier and education domains is as follows :

Supplier <---> Professor
Part <---> course
to teach <--> to supply /* the verbs 'to teach' and 'to supply' are considered analogous */

These analogies at the conceptual level imply the following analogies at the implementation level :

Relation_Supplier <---> Relation_Professor
Relation_Part <---> Relation_Course
Relation_Teach <--> Relation_Shipment

An example of a target query is :

"What are the professors teaching more than two courses which are not taught by profesor P2"

Its equivalent logical form is given below :

```
result([B]) <= setof(C, exists(D, numberof(E, courses(E) &
                                    not(teach(F,E)) &
                                    professor(C) &
                                    teach(C,E) &D> 2, D)), B).
```

Performing the operator α and searching the base library, we found the following base query which best matches the target query :

" What are the supplier supplying more than three parts ?"

The logical form of this query is :

```
result([B]) <= setof(C,  exists(D, numberof(E, parts(E) & supplier(C) &
                                    supply(C,E) &D> 3, D)), B).
```

The implementation of the above query in SQL is :

```
Select Supplier.Supplier_Name
       From Relation_Supplier
       Where  3 <
              (Select COUNT(Shipment.Part)
               From Relation_Shipment
               Where Shipment.Supplier=Supplier.Sup#)
```

Therefore the difference between the base and the target queries consists in only one object which is **not(teach(F,E)).** By modifying the plan corresponding to the base query, we derive the sql form of the target query given below :

```
Select professor.Name
From Relation_Professor
Where professor.name <> prof1 and 2 <
       (Select COUNT(TEACH.Course)
        From Realtion_Teach
        Where TEACH.Professor=PROFESSOR.Key)
```

Chapter 4

Related Work and Comparative Study

4.1 Analogical Reasoning

Humans employ analogical reasoning frequently in their daily life. Analogy is generally used for two different purposes :

- Analogy as a tool for **solving** complex problems
- Analogy as a tool for **comparing** two different situations

Solving complex or unfamiliar problems by using solutions developed for analogous problems requires identifying **similarities** between the two problems, and then **transferring** the solution developed for the base problem to the target problem (the unknown solution). Finding similarities is one of the major problems facing research on analogical learning.

This problem has been studied by [Gentner 1987] as the problem of accessing potential base analogs. When a base analog is identified (i.e. recognized), the process of transferring the known solution to the target problem constitutes the mapping process between the base and the target. As an example, in mathematics this process is frequently used to transfer a problem from one space to another space where the transferred problem is easily solvable (e.g. Using Laplace of Fourier Transforms).

On the other hand, analogy is also suited for **comparing** two different situations. This comparison suggests that the two situations share features obtained by ignoring or reasoning on the differences that may exist between them. As an illustration, [Maiden and Sutcliffe 1991] discussed an example of analogy between a system for theater reservations and a system supporting applications to a university course. Using a graphical representation of these two domains, we can easily identify a set of sharable aspects between the two domains. Among these features, we identify the process of reservation of a seat as analogous to the process of reservation of a place. This analogy suggests that theatergoers and student are also analogous. Another example of analogy proposed by [Miriyala and Harandi 1991], concerns the analogy between a book library and a video library domains. In this case the sharable features are the different primitive specifications performed on the two domains (e.g. primitives borrow, check-out, etc.).

Several approaches have been proposed in the literature to identify sharable features or common aspects between two analogous situations. [Holyoak 1985] proposed the approach based on **pragmatic account** where the matching between base and target is achievable using pragmatic information. Pragmatic approaches tend to emphasize the role of *plans* and

goals in the analogical mapping process. As described in section 3.2, [Carbonell 1986] suggested that a reasoner will always begin by seeking a common goal, then try for a common plan, then a common causal structure and so on. According to Holyoak, the concept of goal in analogical reasoning is critical and influences the matching process indirectly.

In contrast to Holyoak approach, [Gentner 1987] suggests that people use analogies in problem solving by developing mental models of a target domain. The model built is a result of mapping causal models from candidate base domain. This approach focuses on the representation of the base domain. This representation is based on structuring the base domain into a causal model. Using this organization of the base domain, Gentner proposes the structure mapping method, which aims to capture the essential elements that constitutes the analogy and the operations that are computationally necessary in processing analogy. In this case, the analogy is used as a way of focusing on **relational commonalties** retrieved from a set of relations. The retrieved commonalties are independent of the objects in which the set of relations are embedded.

As pointed out in section 2.4, the structure mapping theory is based on the systematicity principle which is considered central to the mapping process. Moreover, systematicity means that the interpretation of an analogy depends not only on predicates from the base domain but also on predicates from the target domain. In this case, a given base-target pair produces a set of matching predicates. Changing members of the pair results in a different set of matching predicates. For this reason, Gentner considers systematicity as a selection constraint defined as follows :

" Among the many possible predicate matches between a given base and target, it favors those that form coherent systems of mutually interconnecting relations [Gentner 1987]."

As an example of using structural mapping approach, [Maiden and Sutcliffe 1991] proposed a framework in which two domains are considered analogous if and only if they belong to a **domain abstraction** model. In this approach, all the common features between the two domains are embedded or abstracted in the abstract domain, which constitutes a **representation** of the analogy.

Another problem facing analogical reasoning research, particularly works related to cognitive science concerns the problem of **interpreting the analogy** [Gentner 1983, Hobbs 1983a, 1983b]. Most of these works attempt to find an explanation of the analogy by means of shared features between a given analogous situations (e.g. Gentner's structure mapping theory). However, the problem with most of these approaches is the fact that analogy is a natural human judgment and therefore all possible interpretations of analogies are quite difficult because of the lack of an exact and complete knowledge about the **context** in which these analogies have been observed. Furthermore, it is easier **to observe** an analogy than to try to interpret it. Indeed, one of the critics formulated against structure-mapping approach is the fact that the base domain is organized around a causal model which is considered as an interpretation of the analogy [Holyoak 1985].

In our approach, we have combined both advantages of the structure-mapping approach and pragmatic account approach. For the structure-mapping approach, we have used a conceptual model which represents the structure of the base domain. For the pragmatic account approach, we have considered that the mapping is aimed to satisfy SAFIR user. Hence, the notion of goal is present during the whole analogical reasoning process. Furthermore, we do not organize the base domain into a structure developed specifically for the analogy; this means we avoid interpreting the analogy. Table 4 summarizes a comparative analysis of related work with respect to the mode of analogy, the purpose of the analogy. and the principle used to derive candidate inferences.

Table 4: Our approach comparatively to other related works

Study	Mode	Purpose	Initial Analogy	Principle
Gentner	Structure driven analogy	Interpretation Analogy	Given object Correspond.	Systematicity
Maiden& Sutcliffe	Structure driven analogy	Specification of application	inferring object correspond.	Abstraction
Myriala& Harandi	Structure driven analogy	Specification by analogy	inferring object correspond.	Difference-based reasoning
Our Approach	Structure and goal driven analogy	Database design and query reuse by analogy	Given an initial analogical observation	Systematicity and abstraction

Table 5 presents a checklist of type of constraints used by different analogical reasoner during the elaboration process (see section 2.7).

Table 5: Types of constraint used by different analogical reasoner during the elaboration process

Study	Semantic	Pragmatic	Structural	Contextual Relevance
Gentner			√	√
Holyoak		√		√
Maiden & Sutcliffe	√	√	√	
Harandi & Miriyala		√	√	√
Our approach	√	√	√	√

4.2 Analogy and Software Reuse

Software reuse is the process of creating software from existing software rather than building systems from scratch [Krueger 1992]. Recently new works in software engineering are conducted to explore the role that analogy can play in reusing software. As an example, [Miriyala and Harandi 1991] present an approach to formalizing specifications by analogy. This approach reuses specification by retrieving a base case analogous to the specification to be formalized. The first difference between their approach and the approach presented in this paper concerns the analogy acquisition process. In [Miriyala and Harandi 1991], the analogy is inferred after the user introduces interactively basic description of the domain used to describe the target specification. In our case, the system starts with an initial

analogy, given by SAFIR-user, representing the correspondence between the objects used in the base domain with the objects of the target domain. From this initial analogy, the system infers the analogy between the relations existing in the database. Another difference with our approach concerns the method used in [Miriyala and Harandi 1991] for handling the difference between the base and the target case. Their approach uses difference-based reasoning to solve this difference. In the so called independent difference, the system adds or deletes the difference (whether if it appears in the base or in the target) only in the target case. In our approach, the difference are accounted for in the base domain. The resulting plan is then transferred to the target without any modifications. However, if the difference could not be expressed in the base (e.g. no analogy stored between the objects in the base domain and the objects appearing in the difference), the system solves the difference in the target domain.

Analogy across domains has been studied by [Maiden and Sutcliffe 1991]. As described in section 2.1 the approach proposed consists to assume that two domains are analogous only if they are both instances of the same **abstract domain** model. An analogy exists if a target and a source domain are analogically **matched** to a domain abstraction. [Maiden and Sutcliffe 1991] tend to support specification reuse through support tools founded on cognitive theories of re-use. In their approach, specification reuse promises more benefit for novice software engineers rather than experts able to retrieve analogous solution from past experiences. In practice, analogy is not well understood when considering surface, lexical properties of the re-usable specification. Successful re-use requires recognition of deeper analogous concepts.

[Bhansali and Harandi 1990] describe a system, called APU, which generates shell scripts for UNIX from a formal problem specification. APU uses derivational analogy to reuse software at both the code and the design level. In order to recognize analogous programs, APU identifies program structure that can be used to solve the target program specification . This process involves a search for a particular object among several similar objects and then some processing on a particular attribute of that object. This means that the retrieval process of a base case is performed at different levels. In our approach, the similarity is performed only on the objects composing the base and target specification. SAFIR retrieves the most analogous base case. The methodology proposed in [Bhansali and Harrandi 1990] consists of stating a general problem, decomposing it into sub-problems, and then solving the sub-problems independently. At this point, the planner of APU uses a backward chaining mechanism to decompose a goal until reaching the primitive level composed by Unix commands. Unlike APU, SAFIR solves a problem indépendently only when a difference between the base and the target is detected.

Chapter 5

Conclusion and Future Work

This thesis described a knowledge-based system in which an inferential mechanism is used to derive software from similar designs contained in a knowledge base. A number of existing systems follow this general approach, in which the inferential mechanism is either backward or forward chaining with respect to a given goal (e.g. [Mostow 1987, Veloso *et al.* 1991). Relatively few systems however incorporate an inference mechanism that uses analogical reasoning in the setting described above. This is surprising given the prominent role that analogy plays in human problem solving in general and in design in particular [Hall 1989].

Our work is significantly different from the few existing systems that use analogy. We tackle a different problem, the definition of a target database and the derivation of database queries from natural language questions, as opposed to the development of operating system macros or the design of information systems. Moreover, the main usefulness of our approach is the fact that it models the behavior of a typical naive database user. This means

that the use of SAFIR presents two main advantages. The first advantage is the transfer of expert knowledge to novice database user. Indeed, the user of SAFIR is constantly exposed to different base domains which represents databases developed by experienced database users. The second advantage concerns the practicality of the approach which is centered around a naive database user as assumed. Such users are generally interested in developing their database applications without having to spend a lot of time on learning database design techniques. An important goal for such users is to query a target database even if the database has been built by applying specific database design techniques in a "shallow" manner.

5.1 Limitations

The approach described in this thesis, even if it has explored new application of analogical reasoning in the context of databases it has some limitations. The most important limitations are described below :

- Limitation of a natural language interface: By using a natural language interface we limit the scope of the expressiveness of the SQL query language. As an example we do not use specific SQL constructs such as CONTAINS, DIVIDE, etc.
- Another limitation of SAFIR concerns the multiple mapping (mapping other than one-to-one) which has not been considered in our approach. As an example the relationship 'treat' could be connected to other objects different from the object patient.

• Unlike [Bhansali and Harrandi 1990] no full planning capability is assumed—and therefore easier to implement. Moreover, we do not attempt to synthesize queries from two partial matches.

5.2 Extensions

SAFIR is a prototype which captures the essentials of the approach presented in this thesis. However, as extensions to SAFIR we distinguish :

• Improving the learning capability in SAFIR : Indeed, when a target query implementation is validated in the target domain, it will be stored for future use. This means that the query reuse process starts first to find an approximate match in the target domain before searching the base domain. With this extension the system is learning from the base domain to construct a library of target query implementation with their correspondent plans.

• Using purpose goals in the acquisition of analogies between base and target domains : This extension suggests that two objects base-target are analogous if their purposes are analogous. As example, if a doctor is analogous to the object professor, then the purpose of doctor "treating patients" is analogous to the purpose of professor "teaching courses".

• Extending the approach in a object-oriented environment: Indeed, object-oriented databases embed concepts from semantic data modeling and currently they are widely used by the database community. The main objective in using object-oriented databases in our approach is to evolve the conceptual model of the base and the target domains.

Currently we are investigating methods of evaluating the approach discussed in this thesis. The objective is to consolidate the proposed methods (i.e. query reuse and database definition). As the objective of the approach is to model the behavior of a typical naive database user, a possible evaluation method to be explored is to train students on SAFIR and to analyze the number of problems solved or unsolved by SAFIR. Another important objective in the evaluation process is to answer the question 'how far the system can help the user ?'.

References

Allen, J. A., Langley, P. and Matwin, S. (1991) "Knowledge and Regularity in Planning", *Stanford Spring Symposium.*

Bhansali, S. (1992) "Software Design by Reusing Architecture", Procs. of *The Seventh Knowledge-Based Software Engineering*, McLean, Virginia.

Bhansali, S. and Harrandi, M. (1990) "The Role of Derivational Analogy in Reusing Program Design", Research Report.

Carbonell, J. G. (1983). "Learning by Analogy: Formulating and Generalizing Plans from Past Experience", in *Machine Learning: An Artificial Intelligence Approach* , J. G. Carbonell. a. T. M. Mitchell and R. S. Michalski, ed., Tioga, Palo Alto, CA, pp. 137-162.

Carbonell, J. G. (1986). "Derivational Analogy: A theory of Reconstructive Problem Solving and Expertise Acquisition", in *Machine Learning: An Artificial Intelligence Approach* , J. G. C. a. T. M. M. R. S. Michalski, ed., Morgan Kaufmann, Los Altos, CA, pp. 371-392.

Date, C. J. (1987). *The SQL standard* , Addison-Wesley Publishing Comp.

Date, C. J. (1990). *An Introduction to Database Systems* , Addison-Wesley.

Desai, B. C. (1990). *Introduction to Database Systems* , West.

Ellman, T. (1989). "Explanation-based Learning: A Survey of Programs and Perspectives", vol. 21, no. 2, pp. 163-221.

Gentner, D. (1983). "Structure-Mapping: A Theoretical Framework for Analogy", *Cognitive Science*, vol. 7, pp. 155-170.

Gentner, D. (1987). "Analogical Inference and Analogical Access", in *Analogica: The first workshop on Analogical Reasoning* , A. Prieditis, ed., Morgan Kaufmann, Pitman, London.

Greiner, R. (1985) "Learning by Understanding Analogies", Research Report, Stanford University, CA, STAN-CS-85-1071.

Groff, J. R. and Weinberg, P. N. (1990). *Using SQL* , Osborn McGraw-Hill, 1990.
Hall, R. P. (1989). "Computatioanl Approaches to Analogical Reasoning: A Comparative Analysis", vol. 39, pp. 39-120.

Hobbs, J. R. (1983a). "Metaphor Interpretation as Selective Inferencing: Cognitive Processes in Understanding Metaphor (Part 2)", vol. 1, no. 2, pp. 125-142.

Hobbs, J. R. (1983b). "Metaphor Interpretation as Selective Inferencing: Cognitive Processes in Understanding Metaphor (Part 1)", vol. 1, no. 2, pp. 17-33.

Holyoak, K. J. (1985). "The Pragmatics of Analogical Transfer", *Psychology Learning and Motivation*, vol. 19, pp. 59-87.

Hull, R. and King, R. (1987). "Semantic Database Modeling: Survey, Applications, and Research Issues", *ACM Computing Surveys*, vol. 19, no. 3, pp. 201-260.

Kodratoff, Y. (1990) "Using Abductive Recovery of failed proofs for problem solving by analogy", Procs. of *Seventh International conference on Machine Learning*, B. W. Porter and R. J. Mooney, ed.

Krueger, C. W. (1992). "Software Reuse", *ACM computing Surveys*, vol. 24, no. 2, pp. 131-183.

LUK, W. S. and Kloster, S. (1986). "ELFS: English language for SQL", *ACM Transactions on DAtabase Systems*, vol. 11, no. 4, pp. 447-472.

Maiden, N. and Sutcliffe, A. (1991) "Analogical Matching for Specification Reuse", Procs. of *Sixth Annual Conference on Knowledge-based Software Engineering*, pp. 101-112.

McFadden, F. R. and Hoffer, J. A. (1988). *Database Management* , The Benjamin/Cummings Publishing Company, Inc.

Miriyala, K. and Harandi, M. (1991) "The Role of Analogy in Specification Derivation", Procs. of *Sixth Annual Conference on Knowledge-based Software Engineering*, pp. 113-125.

Mostow, J. (1987) "Design by Derivational Analogy: Issues in the Automated Replay of Design Plans", Research Report, Department of Computer Science, Rutgers University, MR-TR-22.

Ould-Brahim, H. and Matwin, S. (1992) "Reusing Database Queries in Analogical Domains", Procs. of *The Seventh Knowledge-Based Software Engineering*, McLean, Virginia, pp. 80-89.

Peckham, J. and Maryanski, F. (1988). "Semantic Data Models", *ACM Computing Surveys*, vol. 20, no. 3, pp. 153-189.

Ross, R. G. (1988). *Entity Modelling: Technique and Application* , Database Research Group Inc.

Teorey, T. J. (1990). *Database Modelling and Design* , Morgan Kaufmann.

Thagard, P. and Holyoak, K. (1988) "Analogical Problem Solving: A Constrint Satisfaction Approach", Procs. of *Tenth Annual Conference of the Cognitive Science Society*, Montreal.

Veloso, M. M. and Carbonnell, J. G. (1991) "Learning by Analogical Replay in Prodigy: First Results", Procs. of *EWSL-91*.

Warren, D. H. D. and Pereira, F. C. N. (1982). "An Efficient Easily Adaptable System for Interpreting Natural Language Queries", *American Journal of Computational Linguistics*, vol. 8, no. 3-4, pp. 110-122.

Winston, P. H. (1980). "Learning and Reasoning by Analogy", *Communications of the Associations for Computing Machinery*, vol. 23, pp. 689-703.

Winston, P. H. (1982). "Learning New Principles from Precedents and Exercices", *Artificial Intelligence*, vol. 19, pp. 321-350.

Appendix A :

Script of a Session,

Query Reuse by Analogy

```
| ?- SAFIR.

SAFIR System : Analogy-Driven Reuse of DAtabase queries
Ottawa Machine Learning Group
(c) 1992

Experiment on the Mountain View Database and the Education Database

Entering SAFIR top level...

SAFIR> list analogy.

Analogies at the conceptual level :

          analogy(entity_professor,entity_doctor)
          analogy(relationships_teach(a,b),relationships_treat(c,d))
          analogy(entity_course,entity_patient)

Analogies at the implementation level :

          analogy(relation_PROFESSOR,relation_DOCTOR)
            DOCTOR.name -> PROFESSOR.name

          analogy(relation_TEACH,relation_TREAT)
            TREAT.Patient -> TEACH.course
            TREAT.Doctor  -> TEACH.professor

          analogy(relation_COURSE,relation_PATIENT)
            PATIENT.Key -> COURSE.name

SAFIR> describe databases.

            ******************************************
            *      Description of the two Databases      *
            ******************************************

          Describing the Education database : Target Domain

Object at the IMPLEMENTATION level (TARGET DOMAIN) :
```

Relation PROFESSOR

ID	Name	Spec	Level	Grad
1225	prof1	ML	underg	prof
1226	prof2	DS	underg	grad/Msc
1227	prof3	AI	underg	underg
1228	prof4	NT	underg	prof
1229	prof5	DB	underg	grad/Phd

Relation TEACH

Prof	Course	level	Session
prof1	csi2511	underg	F/92
prof2	csi2512	underg	F/92
prof3	csi5538	grad	W/93
prof4	csi4531	underg	W/93
prof5	csi3033	underg	S/93

Relation COURSE

Name	level	Session
csi2511	underg	F/92
csi2512	underg	F/92
csi5538	grad	W/93
csi4531	underg	W/93
csi3033	underg	S/93

Describing the Mountain View database : Base Domain

Object at the IMPLEMENTATION level (BASE DOMAIN) :

Relation DOCTOR

NSS	Name	Spec	Hospital	Grad
I1275	ss1	CHER	GEN	GEN
I1276	ss2	DERM	HULL_GEN	PROF
I1277	ss3	OPTH	LOC	INTERN
I1278	ss4	OBST	MONT	GEN
I1279	ss5	CARD	CHU	PROF

Relation TREAT

Doc	Patient	Lab	Date
ss1	p2511	MU	11/10/92
ss2	p2512	LU	24/04/92
ss3	p5538	CH	30/12/92
ss4	p4531	RS	15/05/92
ss5	p3033	SM	20/01/92

Press Return to continue...

Relation PATIENT

Key	Name	Date
p2511	Robert/C	11/10/92
p2512	Jean/M	23/04/92
p5538	Patrick/O	30/12/92
p4531	Dan/K	15/05/9
p3033	Cecile	20/01/9

SAFIR> acquire queries in the base domain.

Acquiring queries in the base domain,queries are stored in the file test_bd
Base domain is the Mountain View Database...

Queries_Base_Domain> what are the patients treated by ss1 ?

```
Parse: 33sec.
whq
   _ 1
   s
      np
         3+plu
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+plu
            np_head
               det(the(plu))
               []
```

```
            patient
         reduced_rel
            _ 2
            s
               np
                  3+plu
                  wh(_ 2)
                  []
               verb(treat,passive,inf,[],pos)
               []
               pp
                  prep(by)
                  np
                     3+sin
                     name(ss1)
                     []
   []


Storing the logical form of the query....
Query stored in the file test_bd..

Stored Query: 50sec.

 Query  :

answer([B]) :-
   B = setof C
     patient(C)
   & treats(ss1,C)

Queries_Base_Domain> what are the doctors treating more than two
patients ?


Parse: 17sec.
whq
   _ 1
   s
      np
         3+plu
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+plu
            np_head
               det(the(plu))
               []
```

```
            doctor
         reduced_rel
            _ 2
            s
               np
                  3+plu
                  wh(_ 2)
                  []
               verb(treat,active,inf,[prog],pos)
               arg
                  dir
                  np
                     3+plu
                     np_head
                        quant(more,nb(2))
                        []
                        patient
                     []
               []
      []


Storing the logical form of the query....
Query stored in the file test_bd..


 Query  :

answer([B]) :-
   B = setof C
     exists D
       D = numberof E
         patient(E)
       & doctor(C)
       & treats(C,E)
     & D>2

Queries_Base_Domain> what are the doctors treating patients treated
by ss2?


Parse: 33sec.
whq
   _ 1
   s
      np
         3+plu
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
```

```
      dir
      np
         3+plu
         np_head
            det(the(plu))
            []
            doctor
         reduced_rel
            _ 2
            s
               np
                  3+plu
                  wh(_ 2)
                  []
               verb(treat,active,inf,[prog],pos)
               arg
                  dir
                  np
                     3+sin
                     name(patients)
                     []
               []
         reduced_rel
            _ 3
            s
               np
                  3+plu
                  wh(_ 3)
                  []
               verb(treat,passive,inf,[],pos)
               []
               pp
                  prep(by)
                  np
                     3+sin
                     name(ss2)
                     []
   []


Parse: 850sec.
whq
   _ 1
   s
      np
         3+plu
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
```

```
np
   3+plu
   np_head
      det(the(plu))
      []
      doctor
   reduced_rel
      _ 2
      s
         np
            3+plu
            wh(_ 2)
            []
         verb(treat,active,inf,[prog],pos)
         arg
            dir
            np
               3+plu
               np_head
                  generic
                  []
                  patient
               reduced_rel
                  _ 3
                  s
                     np
                        3+plu
                        wh(_ 3)
                        []
                     verb(treat,passive,inf,[],pos)
                     []
                     pp
                        prep(by)
                        np
                           3+sin
                           name(ss2)
                           []
         []
[]


Storing the logical form of the query....
Query stored in the file test_bd..

 Query  :

answer([B]) :-
   B = setof C
     exists D
       patient(D)
     & treats(ss2,D)
```

```
     & doctor(C)
     & treats(C,D)

Queries_Base_Domain> what is the number of doctors?


Parse: 50sec.
whq
   _ 1
   s
      np
         3+sin
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+sin
            name(number)
            []
      pp
         prep(of)
         np
            3+sin
            name(doctors)
            []


Parse: 600sec.
whq
   _ 1
   s
      np
         3+sin
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+sin
            name(number)
            []
      pp
         prep(of)
         np
            3+plu
            np_head
               generic
               []
```

```
            doctor
        []


Parse: 1117sec.
whq
   _ 1
   s
      np
         3+sin
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+sin
            np_head
               det(the(sin))
               []
               number
            pp
               prep(of)
               np
                  3+sin
                  name(doctors)
                  []
      []


Parse: 1684sec.
whq
   _ 1
   s
      np
         3+sin
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+sin
            np_head
               det(the(sin))
               []
               number
            pp
               prep(of)
               np
```

```
                3+plu
                np_head
                   generic
                   []
                   doctor
                []
      []
```

```
Storing the logical form of the query....
Query stored in the file test_bd..
```

```
 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)
```

```
Queries_Base_Domain> how many doctors are treating pat1?
```

```
Parse: 0sec.
whq
   _ 1
   s
      np
         3+plu
         np_head
            quant(same,wh(_ 1))
            []
            doctor
         []
      verb(treat,active,pres+fin,[prog],pos)
      arg
         dir
         np
            3+sin
            name(pat1)
            []
      []
```

```
Storing the logical form of the query....
Query stored in the file test_bd..
```

```
 Query  :
```

```
answer([B]) :-
   B = numberof C
     doctor(C)
   & treats(C,pat1)

Queries_Base_Domain> what is the number of doctors ?


Parse: 50sec.
whq
   _ 1
   s
      np
         3+sin
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+sin
            name(number)
            []
      pp
         prep(of)
         np
            3+sin
            name(doctors)
            []

whq
   _ 1
   s
      np
         3+sin
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+sin
            name(number)
            []
      pp
         prep(of)
         np
            3+plu
            np_head
               generic
               []
```

```
            doctor
         []


whq
   _ 1
   s
      np
         3+sin
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+sin
            np_head
               det(the(sin))
               []
               number
            pp
               prep(of)
               np
                  3+sin
                  name(doctors)
                  []
      []


Parse: 1966sec.
whq
   _ 1
   s
      np
         3+sin
         wh(_ 1)
         []
      verb(be,active,pres+fin,[],pos)
      arg
         dir
         np
            3+sin
            np_head
               det(the(sin))
               []
               number
            pp
               prep(of)
               np
                  3+plu
                  np_head
```

```
                    generic
                    []
                    doctor
                 []
      []
```

```
Storing the logical form of the query....
Query stored in the file test_bd..


 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)

Queries_Base_Domain> how many doctors are treating more than two
patients ?


Parse: 33sec.
whq
   _ 1
   s
      np
         3+plu
         np_head
            quant(same,wh(_ 1))
            []
            doctor
         []
      verb(treat,active,pres+fin,[prog],pos)
      arg
         dir
         np
            3+plu
            np_head
               quant(more,nb(2))
               []
               patient
            []
      []


Storing the logical form of the query....
Query stored in the file test_bd..

 Query  :
```

```
answer([B]) :-
   B = numberof C
     doctor(C)
   & exists D
       D = numberof E
         patient(E)
       & treats(C,E)
     & D>2

Queries_Base_Domain> end acquisition.

End acquisition...

SAFIR> list query base library.
Starting retrieving queries in the base domain...
Opening file Base domain query Library..test_bd..

Reading Query :

                  what are the patients treated by ss1 ?

Correspondent logical form of the query :

Retrieving: 0sec.

 Query  :

answer([B]) :-
   B = setof C
     patient(C)
   & treats(ss1,C)
Press Return to continue...


Reading Query :

                  what are the doctors treating more than two
patients ?

Correspondent logical form of the query :

Retrieving: 0sec.

 Query  :

answer([B]) :-
   B = setof C
     exists D
       D = numberof E
         patient(E)
       & doctor(C)
       & treats(C,E)
```

```
     & D>2
Press Return to continue...


Reading Query :

                  what are the doctors treating patients treated by
ss2 ?

Correspondent logical form of the query :

Retrieving: 0sec.

 Query  :

answer([B]) :-
   B = setof C
     exists D
       patient(D)
     & treats(ss2,D)
     & doctor(C)
     & treats(C,D)
Press Return to continue...


Reading Query :

                  what is the number of doctors ?

Correspondent logical form of the query :

Retrieving: 0sec.

 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)
Press Return to continue...


Reading Query :

                  how many doctors are treating pat1 ?

Correspondent logical form of the query :

Retrieving: 0sec.

 Query  :

answer([B]) :-
```

```
   B = numberof C
     doctor(C)
   & treats(C,pat1)
Press Return to continue...


Reading Query :

                  what is the number of doctors ?

Correspondent logical form of the query :

Retrieving: 0sec.

 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)
Press Return to continue...


Reading Query :

                  how many doctors are treating more than two
patients ?

Correspondent logical form of the query :

Retrieving: 0sec.

 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)
   & exists D
       D = numberof E
         patient(E)
       & treats(C,E)
     & D>2
Press Return to continue...


Finishing reading input query file...

SAFIR> what are the courses taught by prof1 which are not taught by
prof2?


Parsing the query using xchat-80 System....
```

```
Target Query in the TARGET DOMAIN:
 Query  :

answer([B]) :-
   B = setof C
     course(C)
   & teaches(prof1,C)
   & \+teaches(prof2,C)

Mapping the target query into the BASE DOMAIN : ALPHA-1(Qt): 0sec.

 Query  :

answer([B]) :-
   B = setof C
     patient(C)
   & treats(D,C)
   & \+treats(E,C)


          Matching Module for SAFIR System
          ================================
     1- Structure Matching
     2- Structure-Object Matching
     3- Conceptual Matching
     4- Quit


Choose a number between 1 and 4 [2]: 2


Attempting to match TARGET QUERY with BASE QUERY :

     what are the patients treated by ss1 ?

 Query  :

answer([B]) :-
   B = setof C
     patient(C)
   & treats(ss1,C)

...Press return to continue....

Starting MATCHING process...

SUCCESS : Matching base and target

     finding a DIFFERENCE from the TARGET QUERY

          Difference is : difference(\+treats(T3,U3))
```

```
          Base Query modified :
body(patient(U3),treats(ss1,U3),\+treats(T3,U3))

          Target Query  :
body(patient(U3),treats(T3,U3),\+treats(T3,U3))

Press Return to continue...


Attempting to match TARGET QUERY with BASE QUERY :

     what are the doctors treating more than two patients ?

Retrieving Time: 1sec.

 Query  :

answer([B]) :-
   B = setof C
     exists D
       D = numberof E
         patient(E)
       & doctor(C)
       & treats(C,E)
     & D>2

...Press return to continue....

Starting MATCHING process...

FAILURE : Matching Base and target

Attempting to match TARGET QUERY with BASE QUERY :

     what are the doctors treating patients treated by ss2 ?

Retrieving Time: 1sec.

 Query  :

answer([B]) :-
   B = setof C
     exists D
       patient(D)
     & treats(ss2,D)
     & doctor(C)
     & treats(C,D)

...Press return to continue....

Starting MATCHING process...
```

```
FAILURE : Matching Base and target

Attempting to match TARGET QUERY with BASE QUERY :

     what is the number of doctors ?

Retrieving Time: 1sec.

 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)

...Press return to continue....

Starting MATCHING process...


Attempting to match TARGET QUERY with BASE QUERY :

     how many doctors are treating pat1 ?

Retrieving Time: 1sec.

 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)
   & treats(C,pat1)

...Press return to continue....

Starting MATCHING process...


Attempting to match TARGET QUERY with BASE QUERY :

     what is the number of doctors ?

Retrieving Time: 1sec.

 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)
```

```
...Press return to continue....

Starting MATCHING process...


Attempting to match TARGET QUERY with BASE QUERY :

     how many doctors are treating more than two patients ?

Retrieving Time: 1sec.

 Query  :

answer([B]) :-
   B = numberof C
     doctor(C)
   & exists D
       D = numberof E
         patient(E)
       & treats(C,E)
     & D>2

...Press return to continue....

Starting MATCHING process...

          Selecting the base query :

Retrieving Time: 0sec.

 Query  :

answer([B]) :-
   B = setof C
     patient(C)
   & treats(ss1,C)


Press Return to continue...

          Retrieving the corresponding plan to the selected
          base query

Calling SAFIR Planner...

Press Return to continue...

          Entering SAFIR Planner.....


Processing the  Difference : add(\+treat(v2,p))
```

```
Starting planning on the difference : \+treat(v2,p)

looking for an implementation SQL...

link variable p to the previous object patient(C3)

Find the link operator for object treat(v2,p) translating the
operator to not att_PATIENT.key in
 Find the attribute corresponding to the variable v2 in the object
treat(v2,p)
 The logic variable T3 corresponding to the implementation var s_V2
 from the object treat(v2,p) corresponding to the relation
relation_TREAT
Construct constraint from object treat(v2,p) Imp_var / Attribute =
Var
 Find the attribute corresponding to the variable v2 in the object
treat(v2,p)
 Find the attribute corresponding to the variable p in the object
treat(v2,p)
 The logic variable T3 corresponding to the implementation var s_V2
 from the object treat(v2,p) corresponding to the relation
relation_TREAT
Construct constraint from object treat(v2,p) Imp_var / Attribute_1 =
Imp_2 / Attribute_2

link variable p to the previous object patient(C3)
 Find the attribute corresponding to the variable p in the object
treat(v2,p)

Apply the retrieved attribute..att_TREAT.PATIENT
Apply insert_sql with the sql_part :

Press Return to continue...

Add a link : not att_PATIENT.key in

Select      att_TREAT.PATIENT
from    relation_TREAT
where
[s_V2/att_TREAT.PATIENT=p1/att_PATIENT.key,s_V2/att_TREAT.Doctor=v2]

Press Return to continue...

          Adding the SQL form of the difference to the existing
          SQL base query

               select relation_PATIENT*key
               from  relation_PATIENT
               where relation_PATIENT*key IN
                  select relation_TREAT*patient
                  from   relation_TREAT
                  where relation_TREAT*doctor = ss1
```

```
                        and not relation_TREAT*patient in
                    . select relation_TREAT**patient
                      from   relation_TREAT**
                      where relation_TREAT**patient =
relation_PATIENT*key
                            and relation_TREAT*doctor = ss2

Press return to continue...


          Mapping the new SQL base query into the target domain...

               select relation_COURSES*name
               from  relation_COURSES
               where relation_COURSES*name in
                  select relation_TEACH*course
                  from   relation_TEACH
                  where relation_TEACH*prof = PROF1/gen_Var/s_V1
                        and not relation_TEACH**course IN
                     select relation_TEACH**course
                     from   relation_TEACH**
                     where relation_TEACH**course =
relation_COURSE*name
                           and relation_TEACH*prof =
PROF2/gen_Var/s_V2

 Entering SAFIR top level

SAFIR> bye.
Cheerio.
```

Appendix B

Example of a Plan

```
% Plan Library representing the derivations of queries in the base domain
% File : plan_query can be changed in a given name by using
% change_library_command
% Updated : 05/06/1992

 :- op(200,yf,[in,constraint]).
 :- op(130,fy,[not,select,from,where]).
 :- op(200,yfx,[/,=,suchthat,with]).
 :- op(200,xfy,[===>, *, with]).


plan(query_bd_1,(
        problem_type(solve),
        query_specification(result(p),
                (patient(p),treat(v1,p),name(p))),
        variable_implementation((
                isa(p,object,patient),
                isa(p1,tuple_variable,relation_PATIENT),
                isa(p1(name), attribute,relation_PATIENT))),
        steps((st(patient(p),treat(v1,p),name(p)),
              st(entity(patient(p)),relationships(treat(v1,p)),
                    feature(name(entity(patient(p))))),
              st(p1(name) suchthat
                  relation_PATIENT(p1) with relation_TREAT(s)
                        constraints   s(att_1)=v1 ,
                                      s(att_3)=p1(key)))),
        transformations((tr(patient(p),entity(patient(p)),
                                relation_PATIENT(p1)),
                        tr(treat(v1,p),relationships(treat(v1,p)),
                                relation_TREAT(s)),
                        tr(name(p,c),feature(entity(patient(p))),
                                isa(p1(name),attrib*patient)))),
        constraint_justification((
           ct((relationships(treat(v1,p)),relation_treat(s),constant(v1))
           ===> s(att_1)=v1),
           ct2((relation_treat(s),relation_patient(p1),
                isa(s(att_3), patient_attribute))
           ===> s(att_3)=p1(key)))),

        difference(add(\+treat(v2,p))),
        sql_form(select relation_PATIENT*name
                        from  relation_PATIENT
                        where relation_PATIENT*key in
                         select relation_TREAT*patient
```

```
from   relation_TREAT
where relation_TREAT*doctor = v1)))
```