

SPHINX: Schema Integration by Example

Francois Barbançon and Daniel P. Miranker

{francois, miranker}@cs.utexas.edu

Department of Computer Sciences

University of Texas

1 University Station C0500

Austin, TX 78712-0233

Abstract

The Internet has instigated a critical need for automated tools that facilitate integrating countless databases. Since non-technical end users are often the ultimate repositories of the domain information required to distinguish differences in data types, an effective solution must integrate simple GUI based data browsing tools and automatic mapping methods that eliminate the requirement for a technical user to supervise the process.

We develop a meta-model of data integration as the basis for absorbing feedback from an end-user. The schema integration algorithm draws examples from the data and learns integrating view definitions by asking a user simple yes or no questions. The meta-model enables a search mechanism that is guaranteed to converge to a correct integrating view definition without the user having to know a view definition language such as SQL or SchemaSQL, or even having to inspect the final view definition. We show how data catalog statistics, normally used to optimize queries, can be exploited to parameterize the search heuristics and improve the convergence of the learning algorithm.

1. INTRODUCTION

Integrating data from a given set of databases, implicitly requires making choices about the nature and validity of potential semantic relationships between elements in the database schemas. When the semantic structure of the databases is fully known, it is possible to derive view definitions and schema mappings over those databases, which are semantically correct (Haas et al. 1997). On the other hand, omissions, or uncertainties about underlying semantic properties of the data can result in several competing view definitions, each of them corresponding to a different interpretation of the underlying data representation. We postulate that end-users, through domain expertise, intuitively possess the necessary knowledge to discriminate between competing interpretations.

Many problems in heterogeneous database integration can already be resolved by encompassing user interaction in a point and click interface. This is the case with the specification of attribute mappings in the schema mapping problem, and with building synonym correspondences and unit conversions for a data dictionary. This is not the case for schema integration and data deriving from more than one table. In addition to the above, schema integration requires the specification of query-operators including JOIN and/or higher-order query constructs (Kent, 1991) that have not admitted simple GUI interfaces. We propose a prototype system named Sphinx, designed to extract this knowledge without requiring any abstraction skills from the user.

The reader may recognize that the Clio system (Miller et al. 2000) entails a similar problem definition and, Sphinx like Clio will combine both user interface and machine learning techniques. Clio’s learning component examines possible join paths in the query graph and ranks possible view definitions by estimating their likelihood. Clio then uses data examples to help the user decide between alternative mappings. The user examines a set of illustrative examples as well as the output of the system for the top-ranked view definitions. The user examines those examples and renders judgment as to whether it has converged on a correct transformation. If it hasn’t, the user can provide further assistance through an interaction with the system, based on a range of operators which help guide its convergence to the correct result: “Clio helps the user understand the results of the mappings being formed and allows the user to verify that the transformations that result are what she intended” (Yan et al., 2001).

We developed a meta-model of integrating view definitions enabling an active learning algorithm. This approach offers the following advantages. First, Sphinx is guaranteed to converge to a correct result and reports to the user when it has done so (Clio terminated when the user decided that the generated solution is correct and that no further input is necessary). Second, an active learning system may generate examples that maximize the information gained from the user, reducing the total number of examples viewed by the user, enhancing the user experience. In addition to identifying and ordering heuristics by virtues of our own general knowledge of data integration problems, we further minimize the number of examples a user sees by parameterizing the heuristics with system catalog information and choose examples that optimize information gain in a manner somewhat analogous to using system catalog information to optimize query costs.

The learning component of the Sphinx algorithm uses Version Spaces to represent a meta-model of all possible view definitions (Mitchell, 1977). We detail the Version Space model by considering, syntactically, the possible clauses included in the learned view definition. We place few syntactic limitations on the view definition language. The limitations we do place allow us to contain the size of the search space within tractable bounds. We feel we do so without significantly compromising the language’s practical expressiveness.

The integration of Version Spaces with sample selection from an existing database required us to extend the paradigm. In particular, a new kind of rule for *missing examples* is introduced to deal with ambiguities introduced by various functional dependencies and constraints that occur in schemas and data sets. The use of Version Spaces distinguishes Sphinx from related systems, enabling us to prove that Sphinx is a consistent learner and converges to a correct view definition within a finite number of steps.

We elaborated heuristics based on catalog statistics, motivated for the problem of query discovery and federated view definition. The goal is to decrease the amount of user interaction necessary to explore the space of possible view definitions and converge to a correct result. We evaluate those heuristics over a small set of experimental examples, using Sphinx as a standalone view definition tool, to estimate the burden that will be incumbent upon the user in terms of pursuing the interaction with Sphinx to its successful completion.

We also propose to use Sphinx as a pure user interface, initialized with view definitions generated by an existing automated schema-mapping tool. Our system will provide an interface allowing the user to specify, which of the alternative view definitions under consideration is correct.

After reviewing related work in Section 2, we look at an example of schema integration and illustrate how semantic knowledge of the data plays a role in the federated view definition process (Section 3). In Section 4, we introduce a query-merging algorithm, which takes competing view definitions and allows us to define a search space for Sphinx. In Section 5 we introduce the active learning algorithm. In Section 6, we examine theoretical guarantees for the algorithm and present a proof of correctness and termination. In Section 7, we look at the behavior of Sphinx with practical examples drawn from real-world data integration tasks, and heuristics designed to help converge to a fast result. In Section 8, we discuss the limitations of our approach as well as our prospect for future improvements.

2. RELATED WORK

2.1. Clio and Example-Based Schema Integration

Clio (Yan et al., 2001) is the only system that has design goals and philosophy closely related to Sphinx. The user can visualize the output of the system by browsing both the resulting tables, and by going through a set of “sufficient illustrations”. The learning is unsupervised and the user may choose and construct new examples to submit to the system. This process stops when the user is satisfied the result obtained is correct. The burden of verifying that Clio has learned the correct join path is entirely on the user, and verification is a semi-recursive process: examination of all “sufficient illustrations” is not explicitly sufficient to guarantee Clio has chosen the correct federating query. If, however an inconsistency is discovered by an astute user during the examination of these illustrations, a set of fine-tuning operators allow the user to correct it and bring Clio a step closer to the correct federating query. To that effect Clio requires the user to learn a new paradigm (a set of refinement operators) to help it find the correct result.

The innovations introduced by Clio differentiating it from semi-automated schema matching tools such as SemInt or LSD (see review below) are twofold:

- Clio can learn based on row examples constructed by the user
- Clio seeks to structure the verification process by presenting the query results piecemeal in a set of illustrations. This implicitly recognizes the difficulty of verifying query discovery by simply forcing the user to browse the content of a federated view.

2.2. Version Spaces and Programming by Demonstration

Mitchell proposed version spaces as the set of concepts consistent with a given training set, and delimited by its boundaries with respect to concept *specificity*, a partial order (1978). Haussler studies the PAC properties of version spaces applied to several concepts languages (1988). The computational complexity of the algorithm and the size requirements of tracking boundary sets limit the class of concept languages that version spaces can handle. Alternative representations of version spaces have sought to preserve the original properties of version spaces while extending their applicability. Hirsh proposed instance-based representation of either minimal or maximal boundary sets (1994), and Smirnov proposed a generalized hybrid representation (2001). Other limitations of version spaces include retraction (Idestam-Almqist, 1990), and collapse. The latter is a problem when faced with noisy or inconsistent training data, as well as incomplete search spaces that do not contain the target concept. Smirnov surveys the main body of work in Version spaces, which often abandons the systematic exploration approach and focuses on alternative

representations allowing the system to overcome limitations of version spaces and learn a “best” result like any other inductive approach (Smirnov, 2001). Recent work has also incorporated a version space approach to programming by demonstration (Lau et al., 2003; Lesh and Etzioni, 1996).

2.3.Active Learning

Active learning encompasses the process of selecting the best candidate for inclusion in the training set. Cohn, Atlas and Ladner observed that inductive algorithms consistently require fewer examples when using active learning rather than random selection (1994). Active learning algorithms differ in their approach to selection. Confidence-based (Dagan and Engelson, 1995; MacKay, 1992) and committee-based (Lewis and Catlett, 1994; Seung et al., 1992) selection methods are considered the two main approaches. The benefits of active learning are particularly interesting for learning systems that interface directly with users such as information extraction tools (Muslea et al., 2000; Thompson et al., 1999).

2.4. Semi-automated schema matching tools

Mediator-based architectures to federate heterogeneous databases have drawn a lot of interest (see Abiteboul, et al. 1997; Cluet, et al. 1998; Garcia-Molina, et al. 1997; Haas, et al. 1997; Levy, et al. 1996; Sheth and Larson, 1990; Tomasic, et al. 1996; Vassalos, et al. 1997; Vidal, et al.1998; Yan, et al. 1997 - to name a few). In these systems, the basic assumption is that highly qualified engineers may become domain experts and write the specifications that will drive the data integration. In that line of work, several general-purpose languages for specifying heterogeneous data integration include SchemaSQL and SchemaLog (Lakshmanan, et al. 1996), graph-based ontologies (Mitra, et al. 2000) and XQuery. As shown by Krishnamurthy, Litwin and Kent (1991), such languages must possess higher order features to concisely bridge schematic heterogeneities across data sources (Garcia-Molina, et al. 1997; Kent, 1991; Yan, et al. 1997).

With the maturation of those systems, the problem of generating semantic specifications to federate data sources garnered more attention. Milo and Zohar (1998) theorize that the vast majority of mappings between schema elements in heterogeneous databases are trivial. Those can be derived automatically, and user expertise can be saved for the truly complex mappings. Automated schema matching tools (Castano, et al. 1999; Clifton et al., 1997; Doan, et al. 2001; Li, et al. 2000; Madhavan, et al. 2001; Palopoli, et al. 2000; Scheuermann et al., 1998– to name a few) were developed with the goal of helping an engineer cope with the plethora of domain information, which must be reconciled to produce a mapping between heterogeneous databases. Rahm and Bernstein (2001) propose a taxonomy and a survey of semi-automated schema matching tools.

For example, the LSD system addresses issues of schema matching and unit and dictionary conversion issues (Doan et al., 2001). Whereas Sphinx assumes that the schema matching task is resolved locally by allowing the user to specify a data mapping instance, LSD proposes to examine two databases, their respective schema or ontologies, and use both data and meta-data elements to derive schema correspondences. In addition to 1:1 mappings such as Sphinx assumes, LSD proposes to identify more complex mapping functions such as 1:m mappings as well as some composition or partition functions. Partition or combination operators are necessary when one schema element is mapped to several elements (1:m mappings, m:1 mappings and m:n mappings). Such is often the case with address fields which can be decomposed into smaller fields (street, state, zip ...) or combined into a single large field (Dhamankar et al. 2004). Tools such as LSD assume that two existing source ontologies are

given, and propose to become the expert by using machine learning techniques, in order to discover the correct schema matching. The similarity of schema elements and their eventual matching is driven by a combination of two measures: structural and linguistic similarity. Linguistic similarity is based on a lexical analysis of the data, its domain, as well as and the labels of schema elements. Structural similarity is based on the hierarchical positioning of a schema element in its source ontology.

A further aspect of the automated matching task is record linkage. Whereas most schema matching is concerned with deriving attribute correspondences, record linkage seeks to define object identity across databases. In practice this entails identifying join criteria that are robust to errors and incomplete entries in the database (Grannis et al. 2002, 2003). The issue of handling outer joins as well as inner joins, as it is considered in Clio, is only a small first step in that direction. If attribute correspondence is the vertical matching component of schema integration, record linkage is its horizontal component, and both are clearly necessary in order to define a complete mapping between data elements residing in heterogeneous repositories.

A further aspect of schema integration, are the syntactic data mismatches that must be overcome in schema integration. Domain and unit conversion operators are necessary when data is drawn from heterogeneous domains or different measurement units. Conversions such as ounces to grams, months to quarters, and other sorts of domain mismatch are common in practice. The construction of synonym dictionaries (e.g. ‘Pentium 3’ vs. ‘Pentium III’) or thesauri is characterized by its own technical challenges (see Park, et al. 1995; Takenobu, et al. 1995;) and represents an aspect of database integration outside the scope of our study.

While often having high accuracy, automated tools can never guarantee that a correct mapping has been derived. A database specialist with domain expertise must examine the systems final output to verify correctness. The average user cannot be expected to use the advanced query or semantic modeling languages commonly used to express those mappings. The issues addressed by Sphinx are essentially orthogonal. Sphinx assumes that the user is the ultimate repository of such information and intuition, needed by the integration process. Sphinx has a strong user interface component, and proposes to capture semantics expressed by the user through the manipulation of examples. Semi-automated systems on the other hand assume that this information can be found by examining the data and focus on this process.

3. DEFINING AN INTENTIONAL VIEW

Consider two sources Product Reviews and Supplier Catalog (Figure 1). In order to make information from both those sources simultaneously accessible from a single search interface, we define a federated schema intended to reconcile those two sources.

The federated view, expressed as a query over the source schemas, will provide the roadmap to perform this integration. A mapping of attributes from the source schema to the target schema is immediately apparent. Figure 1 illustrates the correspondence between fields in the source data and the target schema. Such correspondences will be clear to the intended end users. Thus the schema mapping process is natural and intuitive.

This schema mapping is not sufficient to unambiguously define a federated view over the source data. There are four possible joins between **Product Review** and **Supplier Catalog**. Resolving the choice of joins to complete the view definition requires understanding the semantics of the data. There are three main possibilities as the arguments to the join: $[SKU\#] \leftrightarrow [Product_ID]$, $[Name] \leftrightarrow [Name]$ and $[Manufacturer, Name] \leftrightarrow [Manufacturer, Name]$. A

fourth possibility is a right outer join, which will map products listed in the catalog, even if no matching reviews can be found. An additional constraint that should be expressed in the federated view, per the column ‘5-star Catalog’ is that only products with ‘5-star’ ratings in their reviews should populate the target schema. Table 1 illustrates four of many possible view definitions. To determine which of those alternatives is correct is to precisely answer the questions raised above about the nature of the join between **Product Reviews** and **Supplier Catalog**.

<pre>Select SC.Manufacturer, SC.Name, SC.Description, PR.Ed-Review, PC.Price, PC.OrderInfo From Product Review PR, Supplier Catalog SC Where PR.Product_ID = SC.SKU# and PR.rating='5'</pre>
<pre>Select SC.Manufacturer, SC.Name, SC.Description, PR.Ed-Review, PC.Price, PC.OrderInfo From Product Review PR, Supplier Catalog SC Where PR.Name = SC.Name and PR.rating='5'</pre>
<pre>Select SC.Manufacturer, SC.Name, SC.Description, PR.Ed-Review, SC.Price, PC.OrderInfo From Product Review PR, Supplier Catalog SC Where PR.Name = SC.Name and PR.Manufacturer = SC.Manufacturer and PR.rating='5'</pre>
<pre>Select SC.Manufacturer, SC.Name, SC.Description, PR.Ed-Review, SC.Price, SC.OrderInfo From Product Review PR, Supplier Catalog SC Where [(PR.Name = SC.Name and PR.Manufacturer = SC.Manufacturer) OR (SC.Name, SC.Manufacturer) not in (Select PR.Name, PR.Manufacturer From Product Review)] and PR.rating='5'</pre>

Table 1 – Possible View Definitions

4. SEARCH SPACE FOR QUERY DISCOVERY

As a learning algorithm, Sphinx explores a hypothesis space, containing all the possible queries under consideration. The exploration of this search space will yield the correct query. We propose two methods to initialize the search space. The first is a candidate merging method, which takes as input a set of potential view definitions. Each view definition is assumed to be compatible with the same attribute mapping, and could be output by a semi-automated schema-matching tool. Thus we show how to build a search space by merging different queries, but still assuming that

the schema matching problem has already been resolved. We also introduce a second method, which makes no such assumptions, and will build a default search space, using only an initial data mapping provided by the user.

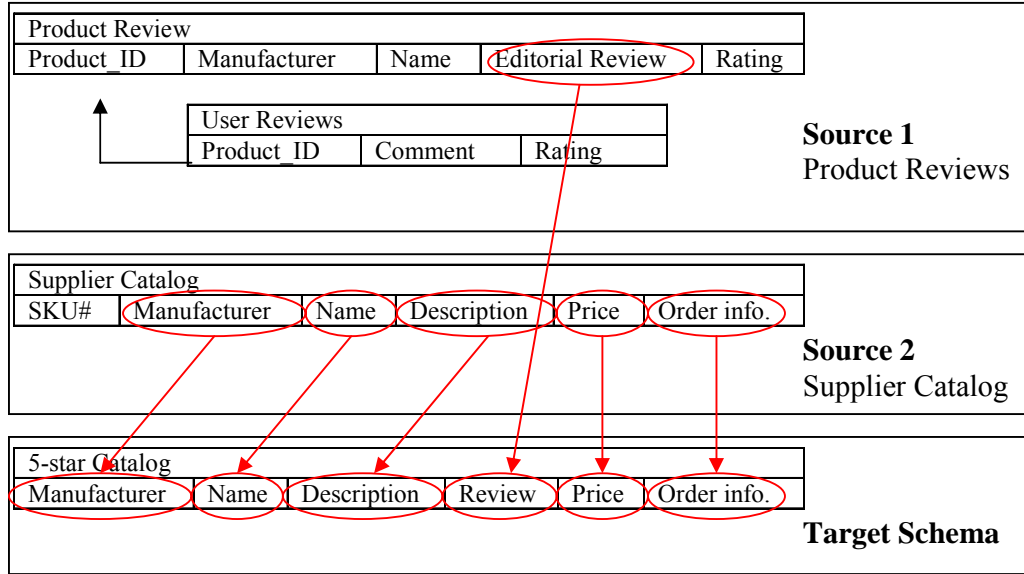


Figure 1 – Federating Data Sources

4.1 Initialization by Query merging

Consider an initial set of view definitions VD_1, VD_2, \dots, VD_n . We write each of those view definitions VD_i as an algebra sentence using projections, selections and a Cartesian product CP_i . We assume that all those view definitions are compatible with a common attribute mapping, such as the one in Figure 1. This means that all agree on which source column to map to each column in the target table. In practice this implies that each of the view definitions can be written with the same formal projection operator to the target view: $\Pi(a_1, \dots, a_t)$, where a_1, \dots, a_t are attributes from source tables.

The correct semantic to describe Cartesian products CP_i is not a set of relations, but rather a bag. Certain relations may appear more than once in a Cartesian product, CP_1, \dots, CP_n . For example, if two mapped attributes originate from the same relation, one view definition could feature a self-join of that relation, while another does not. By using the bag maximum operation (Kent, 1992), each view definition can be rewritten using the maximum Cartesian product $CP_{max} = \max_{bag}(CP_1, \dots, CP_n)$ of all Cartesian products. Assuming that each table features a primary key, this rewriting can be done transparently by adding join predicates in the view definitions, in order to maintain the correct join semantics. For example, if the same relation O_i appears twice in CP_{max} , but self-join is not the desired semantic for some view definition VD_i , it is possible to add a join predicate to VD_i , between the instances of that duplicated relation. Formally if $O_i=O_i'$ then $\Pi(\dots)(O_i) = \Pi(\dots)(\sigma_{O_i.pk=O_i'.pk}(O_i \times O_i'))$.

Thus, assuming that all alternative view definitions under consideration are compatible with the same attribute mapping, we can require that they be rewritten to use the same Cartesian product, and the same projection formula. They will only differ in their selection and join predicates (Table 2). We can then merge those view definitions into a formula parameterized by Boolean

variables F_1, F_2, \dots, F_{pf} (Table 3). Each of those Boolean variables F_i is combined with a selection operator σ_i , such that if F_i is false the selection operator σ_i becomes neutral (the identity function) and if F_i is true the selection operator σ_i applies normally. Each of the original merged view definition can then be expressed within the parameters of this Boolean formula (Table 4).

The resulting formula: $VD(F_1, \dots, F_{pf})$ represents the search space of possible view definitions. The learning algorithm will seek to converge by finding the correct assignments to each Boolean variable (also called feature) F_1, F_2, \dots, F_{pf} . Note that this works even if the queries to be merged contain disjunction, as long as they are expressed in CNF form. Neither do the predicates need to be restricted to equalities. However, our method does not handle disjunction as a general case. We seek to combine the original view definitions on a purely conjunctive basis (Table 4).

$$\begin{aligned}
 VD_1 &= \Pi(a_1, \dots, a_t) \\
 &\quad (\sigma_{1,1} \circ \sigma_{1,2} \circ \dots \circ \sigma_{1, \lambda(1)} \\
 &\quad\quad (O_1 \times O_2 \times \dots \times O_c)) \\
 VD_2 &= \Pi(a_1, \dots, a_t) \\
 &\quad (\sigma_{2,1} \circ \sigma_{2,2} \circ \dots \circ \sigma_{2, \lambda(2)} \\
 &\quad\quad (O_1 \times O_2 \times \dots \times O_c)) \\
 \dots & \\
 VD_n &= \Pi(a_1, \dots, a_t) \\
 &\quad (\sigma_{n,1} \circ \sigma_{n,2} \circ \dots \circ \sigma_{n, \lambda(n)} \\
 &\quad\quad (O_1 \times O_2 \times \dots \times O_c))
 \end{aligned}$$

Table 2 – Alternative View Definitions under Consideration

$$\begin{aligned}
 VD(F_1, F_2, \dots, F_{pf}) &= \\
 &\Pi(a_1, \dots, a_t) \\
 &\quad (F_1 \sigma_{1,1} \circ F_2 \sigma_{1,2} \circ \dots \circ F_{\lambda(1)} \sigma_{n, \lambda(n)} \\
 &\quad\quad F_{\lambda(1)+1} \sigma_{2,1} \circ F_{\lambda(1)+2} \sigma_{2,2} \circ \dots \circ F_{\lambda(1)+\lambda(2)} \sigma_{2, \lambda(2)} \\
 &\quad\quad \dots \\
 &\quad\quad F_{\Sigma\lambda(i)+1} \sigma_{n,1} \circ F_{\Sigma\lambda(i)+2} \sigma_{n,2} \dots F_{pf} \sigma_{n, \lambda(n)} \\
 &\quad\quad (O_1 \times O_2 \times \dots \times O_c))
 \end{aligned}$$

Table 3 – View Definition Formula for Boolean Vector $(F_1, F_2, \dots, F_{pf})$

$$\begin{aligned}
 VD_1 &= VD(F_1=True, F_2=True, \dots, F_{\lambda(1)}=True, False, False, \dots, False) \\
 VD_2 &= VD(False, \dots, False, F_{\lambda(1)+1}=True, \dots, F_{\lambda(1)+\lambda(2)}=True, False, \dots, False) \\
 \dots & \\
 VD_n &= VD(False, \dots, False, F_{\Sigma\lambda(i)+1}=True, \dots, F_{\Sigma\lambda(i)}=True)
 \end{aligned}$$

Table 4 – The original View Definitions are still expressible using the parameterized Formula.

Tables (2-4) – Search Space Initialization by Query Merging

4.2 Default Initialization by Example

A search space can also be initialized using an initial data mapping as shown in Figure 2. Such an initial data-mapping example can be generated by a user with a QBE-like, point and click interface. Constructing such an example does not require advanced abstraction skills from the user, however it is assumed that as a domain expert, the user knows and understands the data sufficiently to produce a correct mapping.

A data-mapping example immediately gives us the proper mapping of attributes from source to target. But it also contains more information because it is grounded with actual data instances from each source, and will enable us to initialize a search space for query discovery.

With the following procedure, we define a parameterized Boolean formula, which will represent the initialized search space:

$$VD(F_1, \dots, F_{pf}) = \Pi(E_1, \dots, E_l) (\sigma_1 \circ \dots \circ \sigma_k (O_1 \times O_2 \times \dots \times O_c))$$

Procedure 4.2.1 (Search Space Default Initialization)

Step 1.

- Introduce a Cartesian product between every tables, which appears in the data mapping example: $O_1 \times O_2 \times \dots \times O_c$.

For each table O_i , introduce variable $\$r_i$.

Step 2.

- Introduce the projection operator necessary to form the target view from the Cartesian product: $\Pi(E_1, \dots, E_l)$, where each E_i is a relational expression using $\$r_1, \dots, \r_c .

Form the Select clause

Step 3.

- Create equality selection predicates $\sigma_1, \sigma_2, \dots, \sigma_k$ for every value, which is part of a row used in the data mapping.

For each table O_i , where o_i is the row from O_i used in the data mapping, introduce for every attribute a_j of O_i , predicate σ : “ $\$r_i.a_j = o_i.a_j$ ”

Step 4.

- Create equality join predicates $\sigma_{k+1}, \sigma_{k+2}, \dots, \sigma_{pf}$ for every pair of values in the data mapping which are in distinct tables and are equal.

For every pair o_i, o_i' of mapped rows such that for some pair of attributes a_j, a_j' : $o_i.a_j = o_i'.a_j'$, introduce predicate σ : “ $\$r_i.a_j = \$r_i'.a_j'$ ”.

We apply this procedure to the example illustrated in Figure 19:

- The Cartesian product is **Product Reviews** x **Supplier Catalog**. The corresponding row variables are PR and SC.
- It follows that the projection operator is $\Pi_{(SC.Manufacturer, SC.Name, SC.Description, PR.Ed.Review, SC.Price, SC.Orderinfo)}$
- The selection predicates are:

$$\begin{aligned} \sigma_1: & \text{“PR.Product_ID=’301-001’”}, \\ \sigma_2: & \text{“PR.Manufacturer=’Hoover’”}, \\ \sigma_3: & \text{“PR.Name=’Supervac 4000’”}, \\ \sigma_4: & \text{“PR.Ed.Review=’A powerful...’”}, \\ \sigma_5: & \text{“PR.Rating=’5’”}, \\ \sigma_6: & \text{“SC.SKU=’301-001’”}, \end{aligned}$$

σ_7 : “*SC.Manufacturer=’Hoover’*”,
 σ_8 : “*SC.Name=’Supervac 4000’*”,
 σ_9 : “*SC.Description=’Vacuum ...’*”,
 σ_{10} : “*SC.Price=’\$49.99’*”,
 σ_{11} : “*SC.Orderinfo=’1-day shipping’*”.

Note that the total number of selection predicates is equal to the total number of attributes of tables **Product Reviews** and **Supplier Catalog**.

- The join predicates are

σ_{12} : “*PR.Manufacturer = SC.Manufacturer*”
 and σ_{13} : “*PR.Name=SC.Name*”.

Note that the data mapping example given by the user immediately excludes a join on attributes *Product_ID* and *SKU#*: the corresponding join predicate is not even generated.

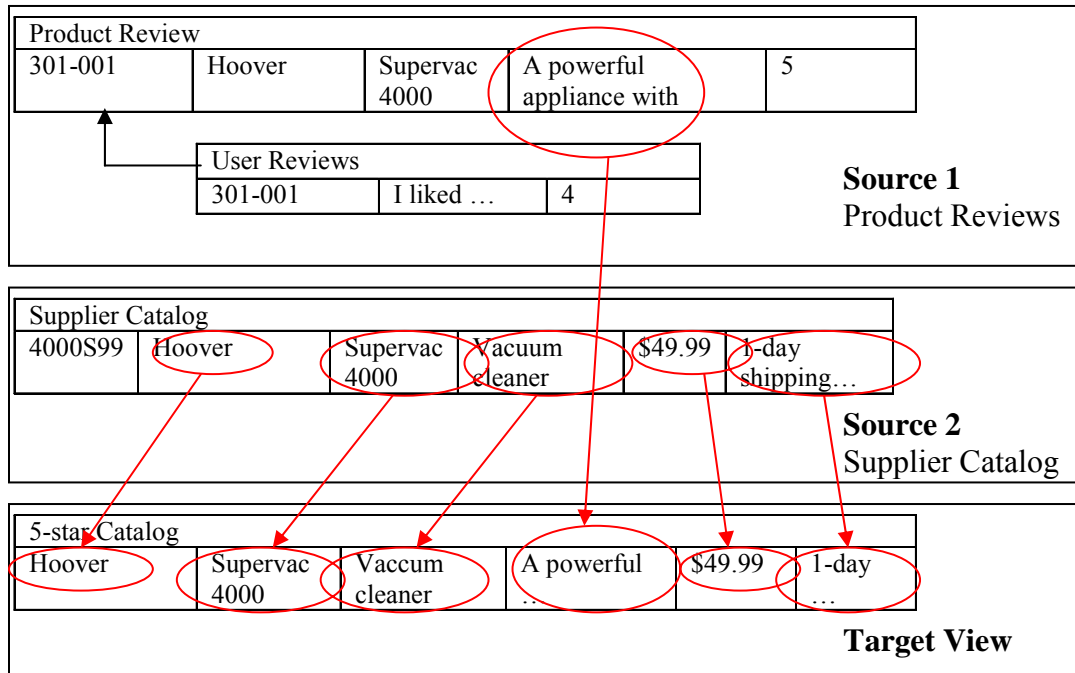


Figure 2 – Data Mapping Example

4.2.2 Completeness of the Default Search Space

This default search space, while fairly general is far from complete. For example, it does not generate view definitions in which the same table appears more than once (self-joins) and only equality predicates are generated. The search space does correspond precisely to the syntactic elements of the language used to express the federating view definitions. The power of the system can be expanded by carefully expanding the syntax and expressiveness of the federating view language and concomitantly adding features to the search space. For example, adding outer joins to the search space would require generating an Outer-Join predicate ($t1.fk = t2.pk$ OR $t1.fk =$

null AND t2.pk = blank) OR (t2.pk = null AND t1.pk = blank)) for each potential join key $[tab1.fk \leftrightarrow tab2.pk]$.

It is possible to make the default search space more complete, by adding extra features such as outer joins, and joins along all possible query paths, but these come at the expense of extra features, and still do not guarantee completeness. The size of the search space doubles with each new feature. But while the search space grows exponentially with the expressiveness of the language, our meta-model representation of the search space remains linear, and that the time required to converge is not exponential.

The current language is expressive enough to be widely effective and represent a large class of challenging queries. Exploration of a language of commercial interest is beyond the scope of this paper.

4.3 Boolean Feature Vector

Either initialization procedure assembles a group of selection and join predicates. The total number of potential predicates being considered is pf . This predicate set: $\{\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_{pf}\}$ determines a search space of federated view definitions, in which both the Select and From clauses are invariants:

- The set of tables O_1, \dots, O_c in the cartesian product determines the From clause as “*From $O_1 \ $r_1, O_2 \ $r_2, \dots, O_c \ r_c* ”
- The projection expression $\Pi(E_1, \dots, E_i)$ determines the Select clause as “*Select E_1, E_2, \dots, E_i* ”

The Where line in our search space consists of a conjunction of predicates, taken from the set $\{\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_{pf}\}$. Hence, the semantics of a federated view definition in our search space will be determined solely by which subset of the set of potential predicates, appears in the Where clause. We introduce the notion of *feature vector* as a Boolean vector of size pf . A *query feature vector* is the practical and unambiguous specification of a query in our search space.

Definition: Query Feature Vector

The *query feature vector* associated with a query q in the search space, and abbreviated $FV(q)$ is a Boolean vector of size pf . Given fixed *Select* and *From* clauses for the search space, if $FV(q) = (q_1, \dots, q_{pf})$, q is the query, and such that $q_i=1$ if and only if filter predicate σ_i appears in the *Where* clause of q .

There are exactly 2^{pf} queries in the Search space, 2^{pf} distinct feature vectors, and we have defined a one-to-one mapping between queries in that space and their query feature vectors. In the rest of this text, we will limit our discussion exclusively to those queries which are in this Search Space or Hypothesis Space, such that for each query q there is a corresponding *query feature vector* $FV(q)$.

There are exactly 2^{pf} queries in the Search space, 2^{pf} distinct feature vectors, and we have defined a one-to-one mapping between queries in that space and their query feature vectors. In the rest of this text, we will limit our discussion exclusively to those queries which are in this Search Space or Hypothesis Space, such that for each query q there is a corresponding *query feature vector* $FV(q)$.

Definition: more specific than, more general than

Let a and b be two feature vectors such that $a = (a_1, \dots, a_{pf})$ and $b = (b_1, \dots, b_{pf})$:

- a is more specific than b (noted $a \geq b$) if and only if $(\forall i: a_i \geq b_i)$
- b is more general than a if and only if a is more specific than b .

5. EXAMPLE-BASED LEARNING WITH VERSION SPACES

The Version Spaces algorithm, presented by Mitchell (1977), is designed to learn the characteristic features of a concept, based on a conjunctive description such as: red, small, round. The features are binary: the concept is either round or not round, it is either small or not small. The features apply to the concept in a conjunctive manner: concepts that either small or round are not considered. The learning takes place by giving the algorithm positive and negative instances of the concept. In this section, we will assume that our goal is to adapt Version Spaces to databases, using a view as the concept, and individual database rows as either positive (the row is in the view) or negative examples (the row is not in the view). The difficulty in adapting Version Spaces to our problem is the observation that not all examples can be classified as either positive or negative. Rows with certain combinations of features will violate certain database dependencies or implicit ‘domain’ related constraints, thus their viability as examples presented to the user for labeling is dubious. Further, rows with certain combinations of features will be theoretically possible, but completely imaginary, making it impossible for our system to present to the user an example, which is grounded in actual data. It will be up to our system to determine, that indeed through dependency constraints or through insufficient data, such is the case for certain combination of features. Ultimately, one may argue that such cases are more akin to negative examples than positive examples. One of the major contributions of this paper is a formal extension of Version Spaces to detect and accommodate such cases, and preserve useful correctness properties for the final view definition.

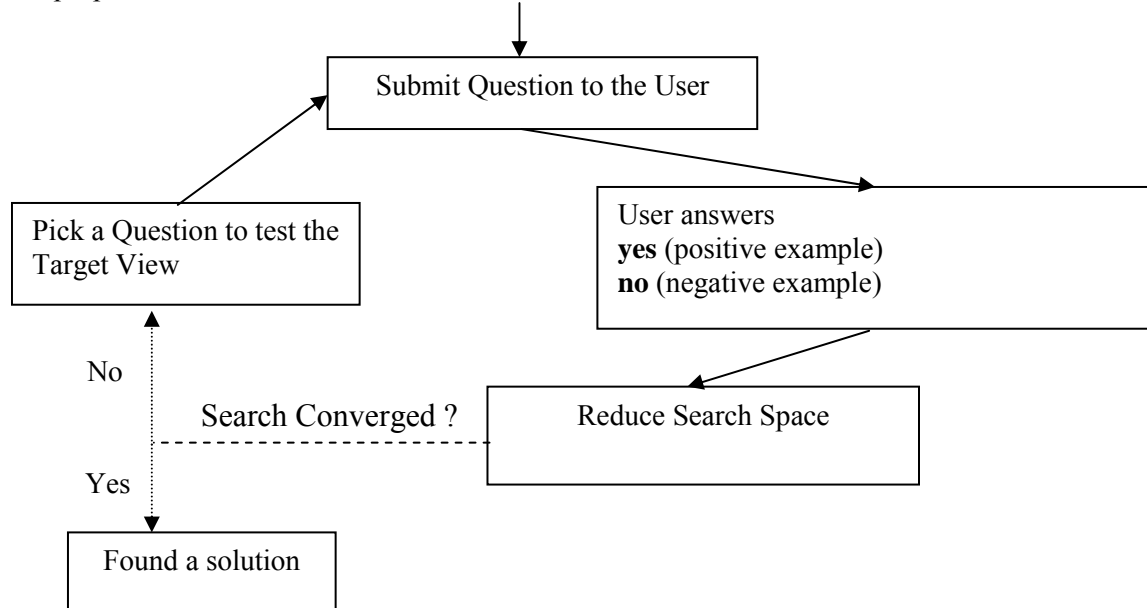


Figure 3 – Question/Answer Interaction

Looking at the problem of query discovery to define views over heterogeneous data, we adopt the terms *target query* and *target view* as the goal concepts for our learning algorithm.

Definition: *Target View, Target Query*

The *target view* is the materialized view that the learning system (Sphinx) is trying to learn from the user. The *target view* is defined by the execution of the *target query* over the source databases.

The user interaction process is analogous to a guessing and elimination game between Sphinx and the user. The goal is for Sphinx to come up with a correct view definition, given a set of question/answer interactions with the user. It will naturally occur to the reader that over a given data source, there may be several queries, which yield the same materialized view as the target query. As such Sphinx cannot always converge on the correct target query. However, Sphinx will identify the entire equivalence class of queries, that correctly materialize the target view over the source data, and will pick the appropriate representative (see Section 6- Lemma 5, Theorem 1).

Figure 3 illustrates the overall question/answer approach to solving the query discovery problem. Note that this whole mechanism is hidden from the user, whose interaction is limited to answering yes or no. The Sphinx learning algorithm performs an iterative loop, whose termination condition is the convergence of the Version Spaces. At each step a feature vector is chosen, and labeled with one of three labels: *positive*, *negative* or *missing*. The missing label is the new label we introduce to extend Version Spaces to learn materialized view definitions over databases. As a result of these repeated steps the target query can be narrowed to a dwindling subset of the search space by the application of three rules. The narrowing subset is commonly called the *space of remaining hypothesis* or *version spaces*. The learning algorithm *converges* when that space is reduced to a single query.

5.1 Version Spaces State

We track the space of *remaining hypothesis* or *version spaces* by its minimal and maximal boundaries with respect to specificity (Mitchell, 1978; Hirsh, 1991). To avoid confusion we introduce two new definitions: the *Version Spaces state* will be the vector containing both boundaries, and the *Query Set* will be the set of queries contained between those boundaries.

Definition: *Version Spaces state*

A Version Spaces state is a pair of items (s, \mathbf{G}) such that:

- s is a query feature vector called the most specific feature vector
- \mathbf{G} is a set of query feature vectors called the most general set.

As shown in Table 5, the Version Spaces is initialized with the initial state (s_0, G_0) , with $s_0 = (1, 1, \dots, 1)$ and $G_0 = \{ (0, 0, \dots, 0) \}$. Thus the initial Query Set $QS(s_0, G_0)$ is the entire search space of legal queries.

Definition: *Query set*

Let (s, \mathbf{G}) be a pair of items such that the first item s , is a query feature vector and the second item \mathbf{G} is a set of feature vectors: $\mathbf{G} = \{g_0, g_1, \dots, g_{mg}\}$.

Query set $QS(s, \mathbf{G}) = \{q \mid q \text{ is more general than } s \text{ and } \exists g_j \in \mathbf{G} \text{ such that } g_j \text{ is more general than } q\}$.

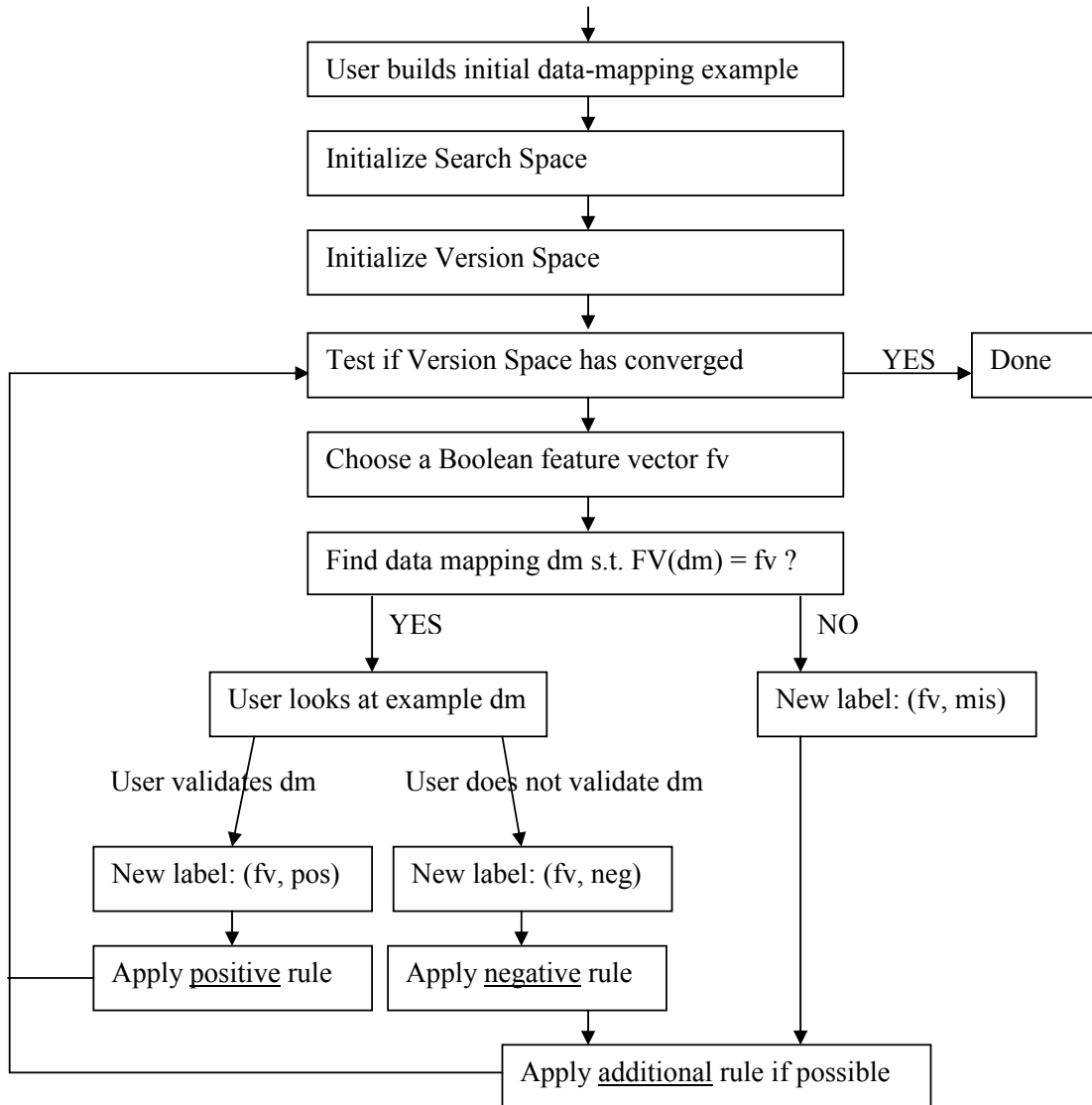


Figure 4 – Chain of Events in the Sphinx active learning algorithm

F₁	F₂	F₃	F₄	F₅	F₆	F₇	F₈	F₉	F₁₀	F₁₁	F₁₂	F₁₃
1	1	1	1	1	1	1	1	1	1	1	1	1

Most Specific Feature Vector

F₁	F₂	F₃	F₄	F₅	F₆	F₇	F₈	F₉	F₁₀	F₁₁	F₁₂	F₁₃
0	0	0	0	0	0	0	0	0	0	0	0	0

Most General Feature Vector Set

Table 5 – Initial Version Spaces state

5.2 Data Mapping

We defined the notion of data mapping instance to formalize the concept of source data coming together to form an object in the target schema. A data mapping is a positive example if the given source data correctly forms a member of the target view.

Definition: *Data Mapping*

A *data mapping* instance dm is an assignment (o_1, o_2, \dots, o_n) of variables $\$r_1, \$r_2, \dots, \$r_n$ in their respective object sets O_1, O_2, \dots, O_n : $dm = (o_1, o_2, \dots, o_n) \in O_1 \times O_2 \times \dots \times O_n$

Definition: *positive, negative data mapping example*

A *data mapping* instance $dm = (o_1, o_2, \dots, o_n)$ is said to form a *positive example* if and only if the target schema object formed by $\Pi_{E_1, \dots, E_k}(o_1, o_2, \dots, o_n)$ is a member of the target view.

If not, dm is said to form a *negative example*.

Figures 2, 5, 6, 8 illustrate the graphical representation for a data mapping. A data mapping can always be represented by showing a set of the data values in the source databases, combining to form an element in the target schema. If the element produced is a member of the target view, the data mapping represents a positive example (Figures 2, 5, 8), if not it represents a negative example (Figure 6).

Just as we did for queries, we can associate a Boolean feature vector with each data mapping instance dm .

Definition: *Example Feature Vector*

For a given data mapping $dm = (o_1, o_2, \dots, o_n)$, the *example feature vector* $FV(dm)$ is defined as $FV(dm) = (\sigma_1(o_1, \dots, o_n), \sigma_2(o_1, \dots, o_n), \dots, \sigma_{pf}(o_1, \dots, o_n))$.

5.3 Rule Definition

We introduce the concept of rules, and the three kinds of rules used in the Sphinx learning algorithm. Formally we define a rule as one of three operators acting on a Version Spaces state. In turn, we will formally define three rules.

Definition: *rule*

A rule is an operator, which takes a Version Spaces state $S = (s, G)$ and a feature vector fv as input and returns a new Version Spaces state $S' = (s', G')$.

5.3.1 Positive rule

The positive rule operator R_p modifies the Version Spaces state by eliminating from the query set those queries that are inconsistent with a given positive data mapping.

Definition: *Positive rule operator*

Let dm be a data mapping such that $FV(dm) = (e_1, \dots, e_{pf})$. The positive rule operator $R_p((e_1, \dots, e_{pf}))$ operates on a Version Spaces state $(s = (s_1, \dots, s_{pf}), G = \{g_1, \dots, g_n\})$, and $g_i = (g_{i,1}, g_{i,2}, \dots, g_{i,pf})$.

$R_p((e_1, \dots, e_{pf}))$: $(s, G) \rightarrow (s', G')$

- $s' = (s_1', \dots, s_{pf}')$ such that

- $e_i = 1 \Rightarrow s_i' = s_i$
- $e_i = 0 \Rightarrow s_i' = 0$
- $G' = G$

Consider the positive data mapping instance dm_1 shown in Figure 5. The mapped values are circled. Because the data in dm_1 is substantially different from the initial data-mapping example in Figure 2, a large number of the potential features predicates do not hold on dm_1 : $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10}, \sigma_{11}\}$. Conversely the following predicates do hold on dm_1 : $\{\sigma_5, \sigma_{12}, \sigma_{13}\}$. Thus the example feature vector for dm_1 is: $FV(dm_1) = (0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1)$. The result of the subsequent application of $R_p(FV(dm_1))$ to the initial Version Spaces state (Table 5) is shown in Table 6. Only the most specific vector is modified: all potential features which are negated in dm_1 see their vector value lowered from 1 to 0. Potential features whose predicates are still fulfilled $\{\sigma_5, \sigma_{12}, \sigma_{13}\}$ suffer no change.

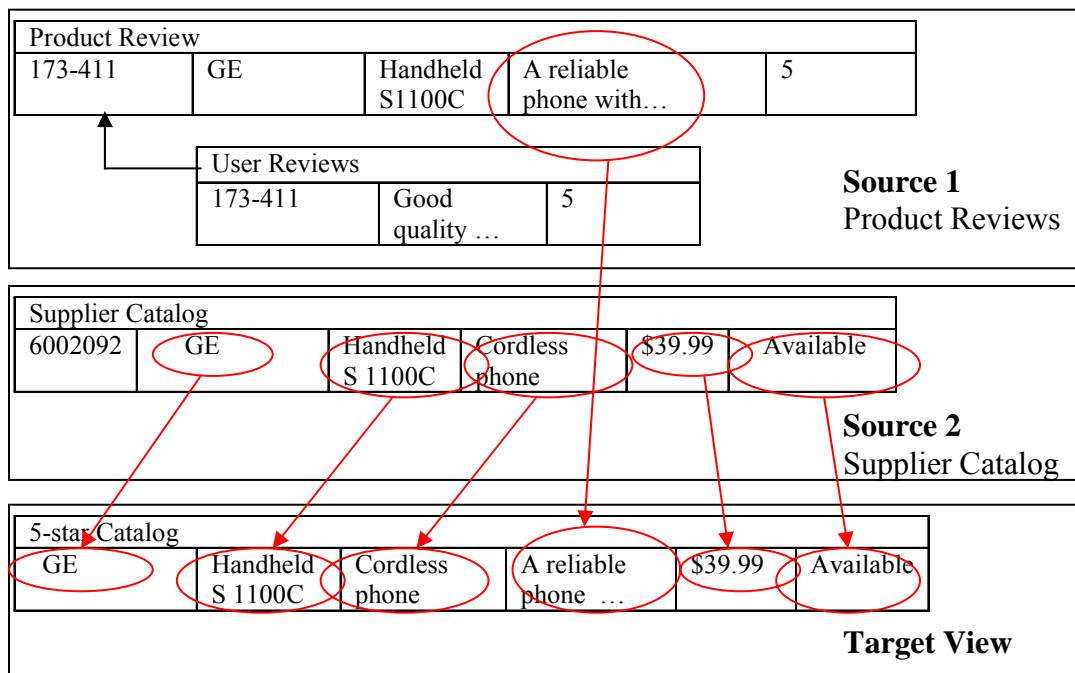


Figure 5 – Positive Data Mapping Example dm_1

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
0	0	0	0	1	0	0	0	0	0	0	1	1

Most Specific Feature Vector

0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---

Most General Feature Vector Set

Table 6 – Applying the *positive rule operator* $R_p(FV(dm_1))$ to the initial Version Spaces state.

The changes shown in Table 6 eliminate from the query set, all the queries with any of the negated predicates $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10}, \sigma_{11}, \sigma_{12}, \sigma_{13}, \sigma_{14}\}$ in their *Where* clause:

since those queries would not produce a target view where dm_1 could be a positive example. This property is formally stated in Lemma 1.

5.3.2 Negative rule

The negative rule operator R_n modifies the Version Spaces state in order to eliminate from the query set those queries that are inconsistent with a given negative data mapping.

Definition: *Negative rule operator*

Let dm be a data mapping, such that $FV(dm) = (e_1, \dots, e_{pf})$.

The negative rule operator $R_n((e_1, \dots, e_{pf}))$ operates on a Version Spaces state $(s = (s_1, \dots, s_{pf}), G = \{g_1, \dots, g_n\})$, with $g_i = (g_{i,1}, g_{i,2}, \dots, g_{i,pf})$.

$R_n((e_1, \dots, e_{pf})) : (s, G) \rightarrow (s'', G'')$

- $s'' = s$
- $G' = \{ (g_{i,1}', \dots, g_{i,pf}') \mid [(\exists p: (g_{p,1}, \dots, g_{p,pf}) \in G) \wedge (\forall j: e_j=0 \Rightarrow g_{p,j} = 0)] \wedge (\exists f: (\forall j \neq f: g_{i,j}' = g_{p,j}) \wedge e_f=0 \wedge g_{i,f}' = 1)] \vee [(\exists p: (g_{p,1}, \dots, g_{p,pf}) \in G) \wedge (\exists f: e_f=0 \wedge g_{p,f} = 1) \wedge (\forall j: g_{p,j} = g_{i,j}')] \}$
- $G'' = \{ (g_{i,1}', \dots, g_{i,pf}') \mid (g_{i,1}', \dots, g_{i,pf}') \in G' \wedge (g_{i,1}', \dots, g_{i,pf}') \leq s' \}$

Consider the data mapping dm_2 shown in Figure 6. It differs slightly from dm_1 , in that the data for source 1 is the same (the product is a GE handheld S1100C), but it is matched with a totally different product from source 2. The feature vector for dm_2 is $FV(dm_2) = (0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$. There are 12 potential features whose predicates are negated in dm_2 : $(F_1, F_2, F_3, F_4, F_6, F_7, F_8, F_9, F_{10}, F_{11}, F_{12}, F_{13})$.

The application of the $R_n(FV(dm_2))$ operator leaves s , the most specific vector unchanged and applies only to G , which has only one member at this stage: $G = \{g_0\}$. The application of $R_n(FV(dm_2))$ happens in two phases:

- In the first phase a new vector g_i' is created from g_0 by changing the bit of exactly one of the 12 features to 1. Since there are 11 features there will be 11 new vectors: $g_1' = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, $g_2' = (0, 1, 0, 0, \dots, 0)$ up to $g_{12}' = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)$.
- In the second phase only those vectors, which are still more general than s will be kept. This eliminates all but g_{11}' and g_{12}' , thus $g_1'' = g_{11}'$ and $g_2'' = g_{12}'$. The final result is shown in Table 7.

5.3.3 Additional rule

The additional convergence rule operator $R_a(p)$ is applied in the negative and missing label branches every time its precondition is met for some potential feature p , $1 \leq p \leq pf$. If the precondition is not met for any p , the Version Spaces state is unchanged and nothing further happens. If it is met for some p , then the operator $R_a(p)$ is applied and the Version Spaces state is modified to reflect the information derived from the precondition being fulfilled.

Definition: *Set O_p*

The set O_p is the set of feature vectors containing a zero at the p^{th} position.

$$O_p = \{ (e_1, e_2, \dots, e_{pf}) \mid e_p = 0 \}$$

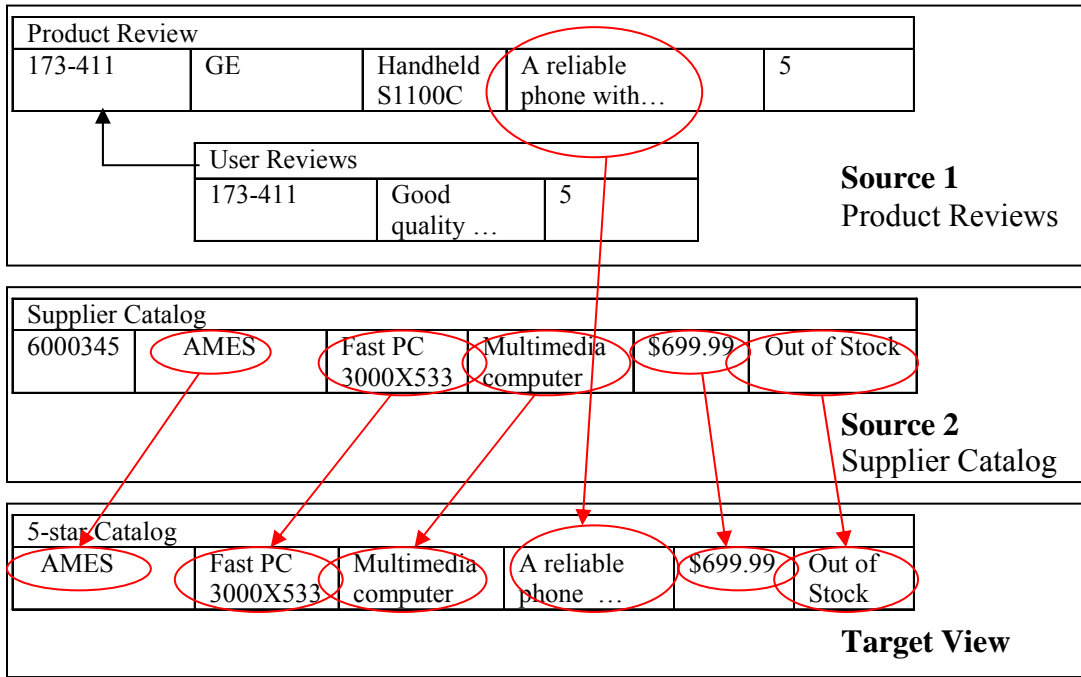


Figure 6 – Negative Data Mapping Example dm_2

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
0	0	0	0	1	0	0	0	0	0	0	1	1

Most Specific Feature Vector

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
0	0	0	0	0	0	0	0	0	0	0	1	0

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
0	0	0	0	0	0	0	0	0	0	0	0	1

Most General Feature Vector Set

Table 7 – Applying the negative rule operator $R_n(FV(dm_2))$ to the Version Spaces state

Definition: *Precondition(p)*

Precondition(p) is met, if for any data mapping dm , $FV(dm) \in 0_p$ implies that dm is a negative example. If there is no data mapping dm , such that $FV(dm) \in 0_p$, then *Precondition(p)* is true.

$(\forall dm : FV(dm) \in 0_p \Rightarrow dm \text{ is a negative example}) \Rightarrow \text{Precondition}(p) \text{ is true}$

Definition: *Additional rule operator (Applied when Precondition(p) is true)*

$R_a(p): (s, G) \rightarrow (s', G')$

- $G' = G$
- $s' = (s_1', \dots, s_{pf}')$, $s_p' = 1 \wedge (\forall i \neq p : s_i' = s_i)$

The original Version Spaces algorithm did not require such a rule. The necessity for the *additional rule* is dictated by the demands of a sample selection system. There will be cases when no data mapping instance for a given feature vector can be found from the existing data sources. Consider predicate σ_5 : $PR.Rating = "5-star"$. Assume all objects in the source have the rating "5-star". Thus all positive data mapping instances validated by the user must contain the rating "5-star". This is insufficient to prove that predicate σ_5 is part of the target view since all negative data mappings must also contain the "5-star" rating. Disproving the presence of σ_5 in the *Where* clause is similarly impossible. Thus, as no examples can be found with a different rating, the system will never be able to determine if the predicate σ_5 should appear or not in the *Where* clause of the target concept.

It should be noted that since all data instances in the source fulfill predicate σ_5 , its presence in the target query does affect the content of the target view. Every query q containing σ_5 in the *Where* clause, has an identical counterpart q' which is similar to q , except for not having σ_5 in the *Where* clause. It is always the case that q and q' materialize the same target view. In that case, applying the *additional rule operator* $R_a(5)$ will reduce the query set by removing for every query q , its counterpart q' .

5.3.4 Impact on Updates

Consider the scenario where the Additional rule operator $R_a(5)$ is applied because no data mapping can be found with a rating other than "5-star". Assume that as the result of an update or a modification of the source data, a rating of "4-star" appears at some point in the future. It will be necessary for Sphinx to re-evaluate the target concept against this new information. Thus every application of the *additional rule operator* should be logged, and a trigger should be introduced as an integrity constraint on the source data.

For each application of an additional rule operator $R_a(p)$, $Precondition(p)$ must become an integrity constraint on the source data. Such an integrity constraint is violated when a data mapping dm appears in the source such that:

- $FV(dm) \in O_p$
- dm has not been labeled a negative data mapping.

Violation of this integrity constraint, and of $Precondition(p)$ must trigger a restart of the learning algorithm at the point where the additional rule operator $R_a(p)$ was applied.

6. CORRECTNESS

As per Section 8, we will discuss the relative completeness of our approach with respect to the limited class of view defining queries based on equality predicates. In this section we look at correctness and prove that the Sphinx learning algorithm, if it converges, it correctly converges to the target concept in the search space. However if the target concept is not in the search space, like all version spaces algorithms, it will eventually yield an empty result. To achieve our proof, we first prove that each rule operator application removes from the space of remaining hypothesis only those queries, which are inconsistent with the data mapping labels. We also prove that the learning algorithm terminates after a finite number of steps.

6.1 Data Mapping and View Mapping Set

The Data Mapping set corresponds to the Cartesian product of the Cartesian sets O_1, O_2, \dots, O_n identified by Algorithm-3. We also define the positive data mapping set as a subset.

Definition: Data Mapping Set DM

The set DM of data mapping instances is formally defined as the Cartesian product of the object sets O_1, O_2, \dots, O_n .

Definition: Positive Data Mapping Set DM^+

Given a query q , such that $FV(q) = (q_1, \dots, q_{pf})$, the positive data mapping set $DM^+(q)$ is the subset of DM which does not negate any of the predicates included in the query q .

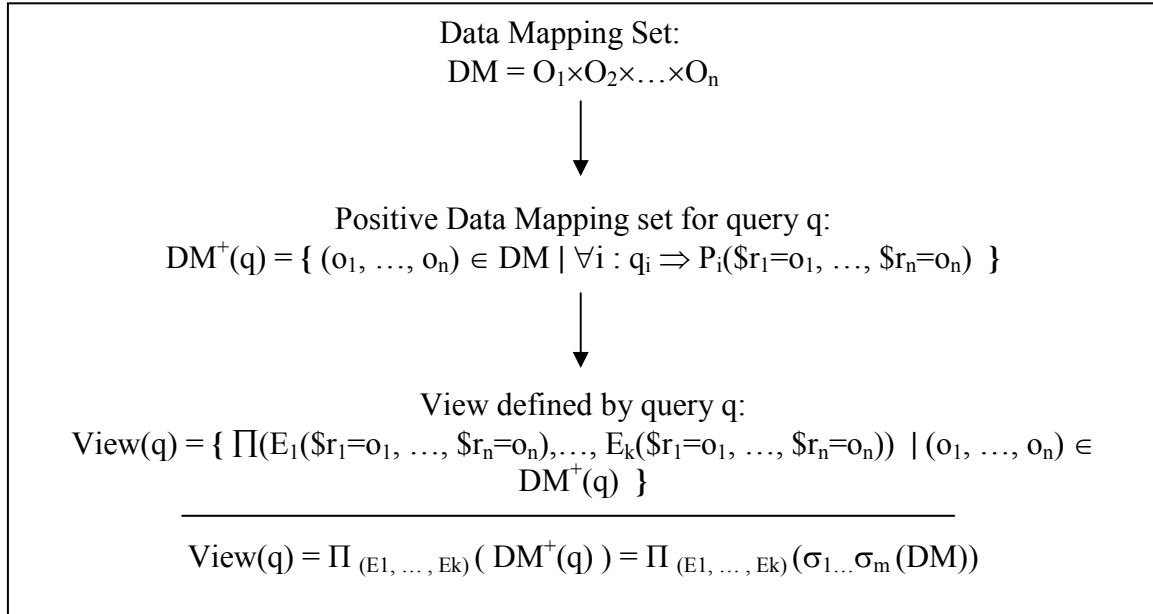


Table 8 – Sets defined by a query

Note that if q_0 is the query with feature vector $FV(q_0) = (0, 0, \dots, 0)$, then $DM^+(q_0) = DM$.
 Note that if a and b are two queries such that a is more specific than b then $DM^+(a) \subseteq DM^+(b)$.

6.2 View Defined by a Query

Any query q in the search space defines a view over the set of source databases. Two different queries q_1 and q_2 may define the same view. A trivial illustration of this is when the source databases are empty.

Definition: View Defined by a Query q

Given a query q , such that $FV(q) = (q_1, \dots, q_{pf})$, we note $View(q)$ the view defined by q .

In the inductive learning model for target query q_t , elements of DM serve as labeled examples: positive examples belong to $DM^+(q_t)$, and negative examples do not belong to $DM^+(q_t)$. Lemma 1 states that a data mapping dm is a positive example for query q if and only if feature vector $FV(dm)$ is more specific than feature vector $FV(q)$.

Lemma 1:

Let dm be a data mapping such that $FV(dm) = (e_1, \dots, e_{pf})$,

Let q be a query such that $FV(q) = (q_1, \dots, q_{pf})$,

$$dm \in DM^+(q) \Leftrightarrow FV(dm) \geq FV(q) \Leftrightarrow \forall i: (e_i \geq q_i)$$

Lemma 1 is merely a restatement in terms of feature vectors of properties expressed in the definition of $DM^+(q)$. Its proof is left to the reader. Lemma 1 allows us to consider labels for positive or negative examples as a property of feature vectors, by stating that the label of a data mapping depends entirely on its feature vector. This observation is stated formally in Lemma 2. Lemma 2 is a direct corollary of Lemma 1.

Lemma 2:

Let dm and dm' be two data mappings such that $FV(dm) = FV(dm')$.
 For any query q , $dm \in DM^+(q) \Leftrightarrow dm' \in DM^+(q)$

6.3 Feature Vector Label

Each feature vector can always be assigned exactly one of three labels: *pos* (positive), *neg* (negative), or *mis* (missing).

Definition: *Correctly labeled pair*

A *labeled pair* is a pair (fv, lbl) , where fv is a feature vector and lbl is one of the three labels $\{neg, pos, mis\}$. Let q be a query, (fv, lbl) is a *correctly labeled pair* with respect to q if and only if the following conditions are met:

- If for all data mapping dm in DM , $FV(dm) \neq fv$, then $lbl = mis$
- If there exists a data mapping dm such that $FV(dm) = fv$ and $dm \in DM^+(q)$ then $lbl = pos$
- If there exists a data mapping dm such that $FV(dm) = fv$ and $dm \notin DM^+(q)$ then $lbl = neg$

Lemma 1 and Lemma 2 guarantee that the three cases in the above definition are mutually exclusive, and that their disjunction is always true.

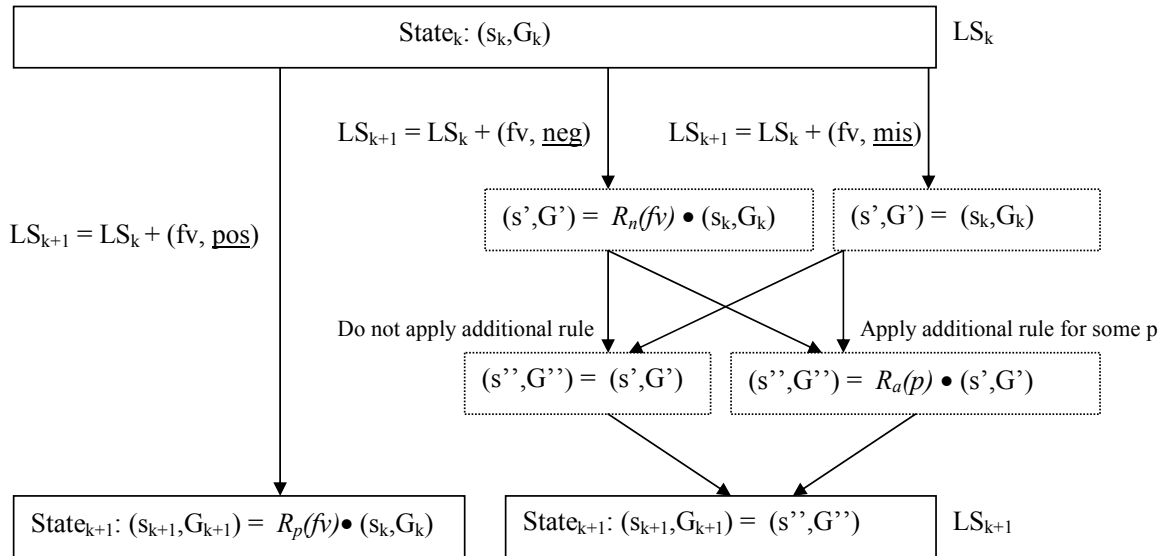


Figure 7 – Sphinx learning algorithm State Transitions

6.4 Learning Algorithm Sequences

Interaction between the user and the system, described in Section 5, results in the user constructing a label sequence, one labeled pair at a time.

Definition: Label Sequence

A label sequence is a sequence of labeled pairs.

Two quantities characterize the state of the Sphinx learning algorithm itself: the Version Spaces state, and a Label Sequence. Figure 7 illustrates the state transition, when a labeled pair (fv, label) is added to the label sequence. The initial state is characterized by vector (s_k, G_k) , and label sequence LS_k . The algorithm branches on the label value, and applies some rules, which reduce the space of remaining hypothesis by operating on the Version Spaces state. As shown there are three different rules and therefore three different kinds of operators exist which operate on the Version Spaces state: the *positive rule* with operator $R_p(fv)$ which is applied in the positive branch, the *negative rule* with $R_n(fv)$ which is applied in the negative branch, and the *additional rule* with $R_a(p)$ which can be applied in both the negative and the missing branch.

Definition: Rule Sequence

A rule sequence is a sequence of rule operators, which are of three kinds: $R_p(fv)$, $R_n(fv)$, $R_a(p)$.

Definition: Rule sequence triggered by a label sequence.

A rule sequence RS is triggered by a label sequence LS, if the rule sequence is the set of actions taken by the Version Spaces algorithm in response to the label sequence LS.

Definition: Version Set

Let RS be a rule sequence. The version set $VS(RS)$ is defined inductively as:

- $VS(\emptyset) = QS(s_0, G_0)$: the empty rule sequence gives a version set equal to the whole search space.
- $VS(RS) = QS(s_k, G_k) \Rightarrow VS(RS + R_x(y)) = QS(R_x(y)(s_k, G_k))$: adding a new rule to the rule sequence is equivalent to applying the rule operator to reduce the space of remaining hypothesis.

6.5 Correctness for Positive rule operator

Lemma 3 states that the *positive rule* operator $R_p(fv)$ removes q from the space of remaining hypothesis (QS) if and only if no data mapping with feature vector fv can be a positive example for q . Thus operator $R_p(fv)$ only removes from the space of remaining hypothesis those queries which are incompatible with the correctly labeled pair (fv, pos) .

Lemma 3:

Let (fv, pos) be a labeled pair, q a query in $QS(s, G)$ and $QS' = R_p(fv)(QS)$:
 $q \notin QS' \Leftrightarrow \forall dm : FV(dm) = fv \Rightarrow dm \notin DM^+(q)$

A proof for Lemma 3 can be found in the Appendix.

6.6 Correctness for Negative rule operator

Lemma 4 states that the operator $R_n(FV(dm))$ removes q from the space of remaining hypothesis (QS) if and only if dm is not a negative example for $View(q)$.

Lemma 4:

Let (fv, pos) be a labeled pair, q a query in $QS(s, G)$ and $QS' = R_n(fv)(QS)$:

$$q \notin QS' \Leftrightarrow (\forall dm : FV(dm) = fv \Rightarrow dm \in DM^+(q))$$

A proof for Lemma 4 can be found in the Appendix.

6.7 Correctness for Additional rule operator

Lemma 5 makes explicit the conditions under which two target queries may define the same target view. In doing so Lemma 5 proves that the *Additional rule operators* only remove from the Version Spaces state redundant target query definition. Lemma 5 guarantees that each view, which could still be the correct *target view*, preserves at least one representative query in the Query Set.

Definition: *Compatible with*

A query q is *compatible with* a label sequence LS if and only if the following properties are true:

- $(fv, pos) \in LS \Rightarrow (fv, pos)$ is a correctly labeled pair with respect to q
- $(fv, neg) \in LS \Rightarrow (fv, neg)$ is a correctly labeled pair with respect to q

Lemma 5(k):

If:

- $LS_k = ((dm_1, l_1), \dots, (dm_k, l_k))$ is a label sequence,
- and RS_k is a rule sequence triggered by LS_k such that $(R_a(p_1), R_a(p_2), \dots, R_a(p_a))$ is the exact subsequence of applications of the *additional rule* in RS_k ,
- and $VS(RS_k) = QS(s_k, G_k)$,
- and q is compatible with LS_k ,
- and q' a query such that $(\forall j \leq a: q_{pj}' = 1) \wedge (\forall i: (\forall j \leq a: i \neq p_j) \Rightarrow (q_i' = q_i))$

Then:

$$q' \in VS(RS_k)$$

A proof of Lemma 5 can be found in the Appendix.

6.8 Correctness and Termination of the Sphinx learning algorithm.

Theorem 1 and Theorem 2 together guarantee eventual convergence of the Sphinx learning algorithm towards a single, correct view definition. Correctness of the result produced by the Sphinx learning algorithm is summed up in Theorem 1. Eventual convergence of Sphinx towards a result is summed up in Theorem 2.

Theorem 1 states that, provided that the target query was in the search space to begin with, a query defining the correct target view can always be found in the *version spaces* (i.e. *space of remaining hypotheses*) maintained by the Sphinx learning algorithm. In particular, if Sphinx has converged to a single query, then that query correctly defines the target view. Theorem 1 is a direct consequence of Lemma 5, and we will leave that proof to the reader. Note that if the search space is incomplete and does not contain the target query, the *version spaces* may collapse and become empty as no solution can be found (Hirsh, 1992; Mitchell, 1978).

Theorem 1:

- Let $v_t = \text{View}(q_t)$ be the target view defined by the target query q_t ,
- let LS be a label sequence such that q_t is compatible with LS ,
- and let RS be the rule sequence triggered by LS :

$\exists q \in VS(RS)$ such that $View(q) = v_t$.

Theorem 2 states Sphinx will eventually exhaust the search space: either converge to a single query or become empty. To discuss the issue of convergence, and prove Theorem 2, it is necessary to introduce the notion of *partial convergence on a potential feature*.

Definition: *Partial Convergence on a Potential Feature*

Assume the Version Spaces state is (s, G) , with $s = (s_1, s_2, \dots, s_{pf})$, $G = \{(g_{1,1}, g_{1,2}, \dots, g_{1,pf}) \dots (g_{k,1}, g_{k,2}, \dots, g_{k,pf})\}$. The Sphinx learning algorithm has partially converged for potential feature F_i , if and only if: $s_i = g_{1,i} = g_{2,i} = \dots = g_{k,i}$.

The reader can verify that if Sphinx has partially converged for all potential features, then it has converged (in the usual sense) and the target query is known.

Theorem 2 states that if each of the 2^{pf} distinct feature vectors is assigned a label, the algorithm is guaranteed to have converged to a solution. The proof figures in the Appendix.

Theorem 2:

Let $v_t = View(q_t)$ be the target view defined by the target query q_t ,
 let LS be a label sequence such that q_t is compatible with LS,
 and such that LS contains 2^{pf} distinct, correctly labeled pairs, one for every possible feature vector instance.
 Let RS be the rule sequence triggered by LS,
 then $VS(RS) = QS(s, G)$ has converged to a single query q , such that $View(q) = v_t$.

7. ACTIVE LEARNING AND SAMPLE SELECTION

Active Learning refers to a process where the system selects examples for the user to label, in order to reduce the cost of labeling unnecessary examples. In this section we look at the problem of selecting the examples that must be submitted to the user.

The number of examples necessary to converge to an answer is an important measure of success for Sphinx and active-learning systems in general. The goal is to converge with a minimal number of examples. The fewer examples, the better the user experience. It should be noted that in an adversarial worst-case scenario, the system could be forced to test $2^{pf} - (pf \text{ choose } 2)$ negative or missing examples, before the *additional learning rule* could be used to converge. Thus the range in performance of the system and the opportunity to introduce heuristics is vast.

7.1 Active Learning as a Search Problem

To explore its search space, Sphinx proceeds incrementally by trying to accomplish a series of intermediate sub-goals. The active learning portion of Sphinx chooses the examples it presents to the user with the purpose of achieving its current sub-goal. Consider the feature vector illustrated in Table 9. All its feature bits are set to 1 except for F_1 . A data-mapping instance with this example feature vector would negate only predicate σ_1 .

F₁	F₂	F₃	F₄	F₅	F₆	F₇	F₈	F₉	F₁₀	F₁₁	F₁₂	F₁₃
0	1	1	1	1	1	1	1	1	1	1	1	1

Table 9 – Example Feature Vector

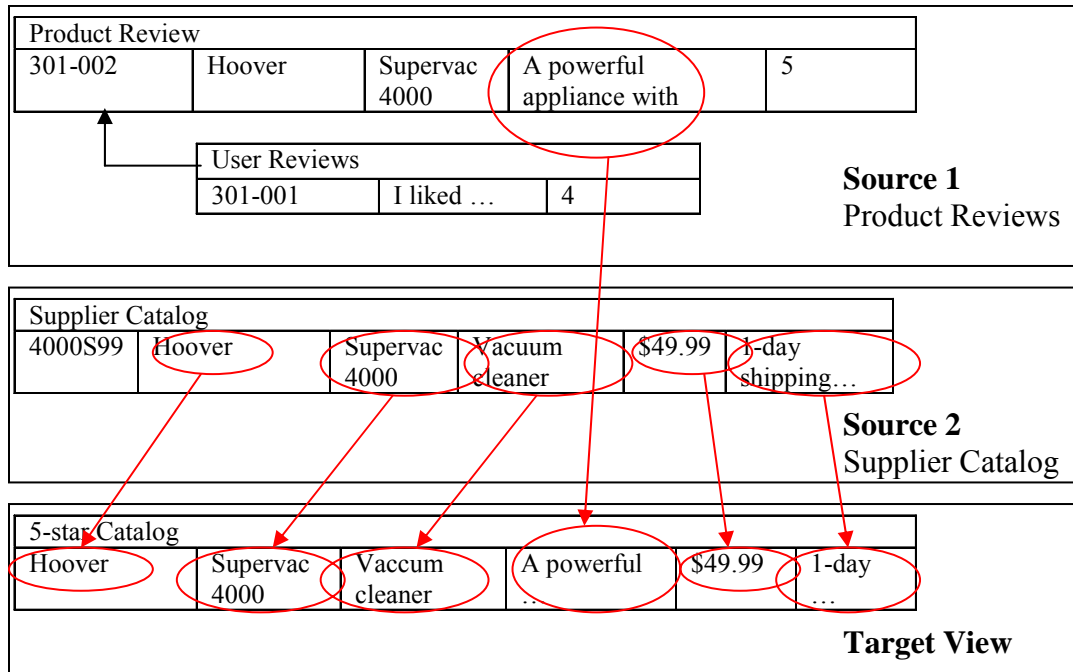


Figure 8 – Data Mapping Instance with the Example Feature Vector shown in Table 9

Consider the data mapping shown in Figure 8. Its example feature vector is equal to the one shown in Table 9. This data mapping is similar in all points to the data mapping shown in Figure 2, except for a minor difference in the value of Product_ID. This minor difference results in the negation of predicate σ_1 .

If the user assigns this data mapping a positive label then Sphinx will know σ_1 cannot be a filter predicate in the target query. Conversely if the user assigns it a negative label then the target query must contain filter predicate σ_1 . Thus, regardless of the actual label value, Sphinx can ascertain whether or not σ_1 belongs to the target query. If the same mechanism is applicable to the 12 other potential features, Sphinx could in this particular instance converge in 13 steps.

7.2 Sub-goals for Partial Convergence

We examine one strategy, which proceeds by setting sub-goals for partial convergence, and incrementally leads to full convergence. This strategy seeks to find data mapping instances for each of the 13 sub-goals shown in Table 10, and then to submit them to the user. To reach these sub-goals, a data mapping instance with example feature vector equal to the sub-goal must be submitted to the user (a query on the source can easily be written to retrieve data mapping instances corresponding to a given feature vector representation). This strategy is particularly attractive, because if the necessary 13 data mappings can be found, regardless of how the user labels them, the algorithm fully converges. However in a practical scenario, using real data sources, none of the data mappings necessary for any of the sub-goals in Table 10, are likely to exist. These sub-goals are too specific: they require data mappings with very restrictive integrity constraints. The reader can observe how unlikely the system is to find a row in **Product Review**, which would produce the mapping shown in Figure 8: if the Product_ID differs from '301-001' to '301-002', it is likely that a lot more description and key fields will differ as well.

Sub-goal 1:	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
	determine if σ_1 must be included in the target query
Sub-goal 2:	(1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
	determine if σ_2 must be included in the target query
...	
...	
Sub-goal 13:	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)
	determine if σ_{13} must be included in the target query

Table 10 – Sequence of sub-goals leading to convergence

Generally, because of functional or accidental dependencies in the data and because many predicates will not be independent (e.g. σ_1 and σ_3) most sub-goals are unreachable and looking for the corresponding data mappings will result in the production of missing labels. Since missing labels do not directly lead to convergence, sub-goals must be chosen carefully.

7.3 Value of Positive Examples vs. Negative Examples

Since a naïve strategy of proving or disproving each individual feature separately (as in Table 10) cannot be seriously considered, it is important to understand that the value of a positive example is much higher than the value of a negative example. Consider a sub-goal, which seeks (to prove or disprove) a group of k filter predicates. Finding a positive data mapping example for that sub-goal, will bring the algorithm closer to convergence: by generalizing the most specific feature vector on all k features, the algorithm automatically reaches partial convergence on those k features. On the other hand a negative data mapping example for the same sub-goal does not bring the algorithm much closer to converging, because when all k features are negated at the same time, it is not the case that partial convergence is achieved on any feature. The nature of the conjunctive language of candidate queries is such, that the *version spaces* admit a least upper bound. Positive examples, by directly altering that least upper bound, eliminate more candidates.

7.4 Sample Selection Heuristics

In this section, we look at heuristics to implement sample selection for the Sphinx system. When looking at establishing facts about the target query using sub-goals, we need to consider two factors:

- missing labels are of little value,
- negative labels are also of little value.

Any strategy wishing to minimize user interaction, must focus on finding at least one, possibly several, examples labeled *positive* by the user. Therefore the basis of our strategy is to separate active learning in two phases. In the first phase Sphinx will exclusively focus on proposing data mappings, which it judges are likely to be positive examples. The goal in the first phase is to converge quickly on as many features as possible. In the second phase Sphinx proposes those data mappings that have no particular reason to be positive or negative. Thus the expected convergence rate in the second phase will be minimal, but with the extent of the search space reduced by the first phase, the number of examples overall can be contained.

7.4.1 Join Feature Bias

A simple analysis shows that the overwhelming majority of the potential features are selection predicates. A selection predicate is created for every attribute in the schema. A join feature is created only when supported by the initial data-mapping example: a relationship exists between two objects in the data mapping.

		Potential Features	Selection Features	Join Features
Healthcare	Query 1	14	14	0
	Query 2	15	15	0
Sports Statistics	Query 3	25	24	1
	Query 4	25	24	1
5 Star Catalog	Query 5	30	28	2
	Query 6	30	28	2
SMD → Base	Query 7	57	57	0
	Query 8	35	33	2

Table 11 – Selection vs. Join Features

An accidental match between a “9.99” as the price \$9.99 and a “9.99” as September 99 is possible, but is an unlikely event given any pair of objects chosen at random by the user. Thus almost all join features observed in the data-mapping example are not flukes and do represent existing semantic relationships. Thus two factors combine here to privilege join features: a pure Cartesian product without a join predicate is a very unlikely operation, and most observed join features between objects belonging to different tables are not accidental.

We elaborate a baseline strategy S_b based on this observation. S_b privileges the search for positive examples in its initial phase by initially never choosing sub-goals negating join predicates. When searching for a sub-goal, strategy S_b looks to negate only a small set of selection predicates (never more than 10). That search is repeated by modifying the set until a positive example is found or the algorithm converges. The small set of negated selection features is chosen with an initial randomization and modified in an incremental search pattern. There is a possibility for backtracking to re-randomize the set, but in the course of running the experiments shown in Table 12, backtracking with S_b occurred only once. Once positive examples have been found, S_b enters its second phase, in which both join and selection predicates will be negated.

It should be observed that in addition to its bias for join features vs. selection features, strategy S_b possesses another built-in bias: it consistently bets that of all the potential features (a large number), only a very small number is likely to actually appear in the *Where* clause of the target query. The latter bias is consistent with our observation that only a small portion of all legal predicates are likely to appear in the target query.

7.4.2 Information Gain Bias

We seek a further bias to predict which predicates are more likely to appear in the cardinality component of the target query. We observe that not all features are equally likely. Consider the following feature predicate: “Name = ‘Supervac 4000’”. It is unlikely to appear in a view defining query since few objects in the source, perhaps only one, will fulfill that predicate, making it useless for any view definition. On the other hand a feature such as “manufacturer/state = ‘TX’” is more likely. A significant proportion of the objects may well fall in the ‘TX’ category and building a view with those might be of use.

We can measure for each selection predicate their information gain. The basic assumption is that the information gain will serve to estimate the likelihood of a predicate. The information gain $IG(\sigma)$ is a function of the filter factor $FF(\sigma)$, and is maximal when $FF(\sigma) = 0.5$.

$$IG(\sigma) = - (|S_1|/(|S_1|+|S_2|))\ln(|S_1|/(|S_1|+|S_2|)) - (|S_2|/(|S_1|+|S_2|))\ln(|S_2|/(|S_1|+|S_2|))$$

$IG(\sigma)$ will yield high scores for predicates on enumerated types, and low scores for predicates on infinite types. We note the information gain metric $IG(\sigma)$, relies on the same catalog statistics used in those query cost models. It has been shown these statistics can be derived even in distributed systems where catalog information is not directly accessible; see (Haas, et al. 1997; Tomasic, et al. 1996). Thus, we are confident we can always rely on such statistics to drive our heuristics.

We introduce a new strategy S_c , a refinement of S_b based on this information gain bias. This strategy does not require likelihood estimations to be accurate. The likelihood function should, above all, cluster predicates into two major categories: the most unlikely predicates (extremely low information gain and infinite domain), and the other predicates (low to high information gain and enumerated domain). This clustering will replace the random process used in S_b . A comparable bias could be introduced to estimate the likelihood of individual join predicates, however as discussed earlier the small number of join predicates precludes the need.

7.5 Experiments

We implemented a prototype system complete with graphic-user interface. This prototype handles data mapping instances presented here, as well as mappings from meta-data elements to data (as in SchemaSQL – Lakshmanan, et al. 1996). This small higher-order generalization allows from a broader range of restructuring queries across schematically disparate sources, without any substantial changes to the overall system.

We chose four domains to experiment with data integration. All of these problems were actual internet database integration tasks conducted under contract, in an ad-hoc fashion by a web services consulting firm. Almost all source data was available only in HTML form, with the source sites wrapped (Baumgartner, et al. 2001; Crescenzi, et al. 2001) to produce structured results. These experiments with Sphinx recreate those schema integration tasks, and are ranked in Table 11, by increasing level of empirical complexity. In the first domain the target queries populate a Healthcare provider directory database. The second domain is based on sport statistics databases. The third domain is the ‘5 Star Catalog’ for electronics and comes from the area of online pricing catalogs for B2B merchandise distributors. A slightly simplified version is used as an illustrative example in this exposition. The last domain involves data migration between two application platforms for Gene Expression Microarray data management. Experimental validation of the Sphinx bias on a set of canonical applications is not possible since no such benchmarks exist in the heterogeneous database integration literature (Florescu et al. 1998). Rather than focus our analysis on a schema integration problem of our own design, like our scientific predecessors we sought anecdotal, real world data integration case studies. The resulting analysis will show that the bias in Sphinx is at least plausible and can successfully address real world problems.

Table 12 gathers some results: each test set includes source databases, a target schema, and two interesting target queries populating different tables of the target schema. The number of features (i.e. Selection and Join predicates) appearing in the *Where* clause of each query is shown (e.g. 1J, 0S: 1 Join and no Selection predicates) and is roughly equal to the number of attributes in

the tables used to define the target view. The number of examples Sphinx requires to reach the target query is averaged over ten runs for each query and shown in Table 12. The decimal averages are a product of random factors present in both heuristics.

Strategy S_b is more successful than S_c for the target queries which do not have a selection predicate in their *Where* clause. The performance of strategy S_b degrades quickly when the target query contains even a single selection predicate. Strategy S_c , shows a more stable behavior, its performance only slowly decreasing when the complexity of the target query increases. This can be attributed to the inability of strategy S_b to differentiate among the selection features, and hence pick the ones, which can be excluded from the target query with high probability.

Table 13 compares the performance of both active learning strategies for Sphinx with two experiments in which Sphinx is hobbled to become a passive learning system. These two experiments do not represent valid strategies but are designed to identify bounds, both lower (oracle) and upper (random) on the number of examples an active learning algorithm may require. In these passive learning experiments, the user carries the burden of constructing data mapping examples as well as labeling them. Sphinx merely indicates when the system has converged to a target query. To measure a lower-bound, an omniscient user, (us), constructed the optimal sequence of examples to converge Sphinx as quickly as possible. To measure an upper-bound a naive user chooses examples at random from the set DM. At each step there is an equal probability of a positive or a negative example being chosen. Each example is picked randomly from its respective population of positive (DM^+) or negative examples ($DM \setminus DM^+$) with equal probability. Unlike the oracle this user is not in a feedback loop with the learning system, and does not know which examples need to be selected in order to reach convergence as quickly as possible.

These experiments show that even with imperfect heuristics, the observed complexity is correlated with the size and complexity of the target query rather than with the number of potential features. We can observe both that the number of required examples spikes when predicates are added to the target query, and that the most simple target queries require a small number of examples, even when the number of potential features is large.

				Strategy S_c			Strategy S_b		
		Query size	Potential Features	Number of Examples			Number of Examples		
				Total	Pos.	Neg.	Total	Pos.	Neg.
Healthcare	Query 1	0J, 0S	14	2.2	2.2	0.0	1.4	1.4	0.0
	Query 2	0J, 1S	15	4.7	2.0	2.7	5.5	2.5	3.0
Sports Statistics	Query 3	1J, 0S	25	4.9	3.7	1.2	2.5	1.5	1.0
	Query 4	1J, 1S	25	4.8	1.8	3.0	11.4	1.6	9.8
5 Star Catalog	Query 5	2J, 0S	30	5.9	3.2	2.7	4.0	2.0	2.0
	Query 6	2J, 1S	30	8.9	2.7	6.2	11.7	2.0	9.7
SMD → Base	Query 7	0J, 0S	57	4.5	4.5	0.0	x	x	x
	Query 8	1J, 0S	35	3.2	2.2	1.0	3.0	2.0	1.0

Table 12 – Number of examples required to converge. Two heuristics compete.

		S _c	S _b	Oracle			Random		
		Total	Total	Total	Pos.	Neg.	Total	Pos.	Neg.
Healthcare	Query 1	2.2	1.4	1	1	0	26	13	13
	Query 2	4.7	5.5	2	1	1	26	13	13
Sports Statistics	Query 3	4.9	2.5	2	1	1	7.5	3.5	3.0
	Query 4	4.8	11.4	3	1	2	22	11	11
5 Star Catalog	Query 5	5.9	4.0	4	1	3	12	5.5	6.0
	Query 6	8.9	11.7	5	1	4	42	21	21

Table 13 – Number of examples required to converge. Sphinx selects examples vs. an oracle vs. random selection.

8. DISCUSSION

To place this work into a larger context we review first what is currently within the scope of our Version Spaces model, and second of our feasibility prototype, Sphinx. We speak to the scope of features and predicates that can be addressed in this model and those that must be addressed in other ways. We characterize the Sphinx approach as an active learner based on the version spaces and incorporating a traditional inductive bias as a sample selection engine.

8.1 EXPRESSIVE POWER OF OUR CURRENT APPROACH

The Version Spaces model used by Sphinx can handle any query, expressed by a conjunction of arbitrary predicates of any kind. Predicates may contain any arithmetic such as ‘ $x.a+3>y.a$ ’, or more complex features such as negation ‘ $x.a\neq y.a$ ’, outer joins ‘ $(x.a=y.a \text{ or } x.a=\text{null} \text{ or } y.a=\text{null})$ ’, disjunction, etc.. Such queries are expressible with the algebraic formula shown in Table 3 and fall within the expressive power of Version Spaces and of the active learning algorithm for which we elaborated formal correctness and termination guarantees. However when we build the search space, we seek to form a new view definition by combining potential predicates on a purely conjunctive basis.

Our current model does not include query features based on attribute correspondences: competing hypotheses are assumed to be compatible with the same given set of correspondences (or harvested from an initial data mapping example – which amounts to compatibility). The issue of identifying such mappings is central to the federating process and is the sole focus of more than one semi-automated tool. However as we noted earlier, the process of specifying and verifying those with an expert user, is fairly straightforward and intuitive through a non-technical point and click interface (see Section 3 and Figures 1, 2). Hence, Sphinx focuses with confidence, on the more delicate issue of example-based expression of semantic joins and higher order features.

In our feasibility prototype, we incorporated a fully automated way of building a search space with a limited class of queries (equality predicates) and with no user intervention. With this initial GUI prototype, we explore how far a fully automated system with almost no user interaction can go, in an area that has traditionally challenged our ability to propose GUIs suited for non-technical users. A more advanced GUI prototype may give knowledgeable users the opportunity to enter additional input in simple pull-down menus or by asking more technical questions. Further, incorporating key generating (or Skolem constant generating) functions,

which would allow the vertical partitioning of source data into separate target views, would pose no difficulty in our model.

As our goals were to push the envelope on a fully automated system, we do not handle unit conversions, or aggregation. In generating a default search space, we explicitly curtailed Sphinx to a limited class of queries with equality predicates. Without additional input from the user, currently generated potential predicates include all legal equality selection predicates on objects involved in the initial data mapping. They also include all legal equality join predicates on those same objects with the exception of self-joins and of a peculiar kind of join, characterized by join paths in the query graph that traverse children of data mapping elements. As shown by Clío, a more exhaustive, though still not complete, generation of all equality selection and join predicates is also possible, by building a query graph and taking into account every possible join path between relations used in the attribute mapping.

We just discussed the limitations of Sphinx in terms of the completeness of its search space: if the target query is in the original search space, Sphinx will find it, but will otherwise fail by exhausting the version space without finding a solution. This basic philosophy was driven to us by the practitioners in the field who provided our application databases: it is okay for a user interface to fail, but it is not acceptable to give an incorrect solution. As noted by Popescu, Etzioni and Kautz, high reliability in user interfaces can be more important than intelligence (2003).

8.2 Adapting the Sphinx active learning approach to a traditional likelihood-biased learning system.

Sphinx incorporates two elements:

- a classic inductive learning task (in this case query discovery) for which we present a motivated quantitative bias
- an active learner, which systematically keeps track of remaining candidates in the version spaces

Typical inductive biases fall into two categories: *likelihood* biases distinguish between different concepts in the search space based on some score or estimation and second, *structural* biases limit the search space to a set that is likely to closely fit the concept. Committee-based sampling methods for active learning tend to derive from the latter (structural bias), whereas certainty-based active learning methods naturally derive from the former (likelihood bias). The Sphinx approach to active learning is suitable to the latter. Sphinx aims to demonstrate how active learning and concept verification can complement a traditional learning method with an inductive bias that is expressed in terms of quantitative likelihood estimations.

A traditional inductive learner concerned with producing a ‘best’ result after a given training set need to be concerned about data dependencies and real world constraints that cause missing examples. Our adaptation of the version spaces approach makes it possible to track the *space of remaining hypotheses* and verify a target concept even in the presence of the data dependencies that will result in the absence of some examples crucial to disambiguating competing queries. However, because of the futility of bias-free learning, the actual learning engine within Sphinx derives from a classic inductive bias converted into an example selection heuristic. This heuristic, if applicable on the test data, allows Sphinx to verify the concept after a small number of examples. If on the other hand the bias is not applicable, Sphinx will fail to converge to a single hypothesis within a reasonable number of examples. Thus as we hoped to demonstrate in some small measure with our experiments, the success of Sphinx is more directly

linked with the effectiveness of the underlying inductive bias and its applicability than with the complexity of the search space. In this respect, this is typical of traditional machine learning problems.

Thus, our goal is for our work with Sphinx to pave the way for adapting existing machine learning algorithms in schema mapping tools with a verification and active learning query verification component based on our modification of version spaces (i.e. three-valued version spaces). In its current form, the search space handled by Sphinx is appropriate for expressing record-linkage and for predicate discovery. We are confident for the reasons outlined previously in this discussion that the same approach can be extended to allow Sphinx to verify attribute correspondences, such as those derived by a tool like LSD. We surmise that adapting semi-automated schema integration tools (see Section 2 for a review of such tools) to verify their result through a non-technical and intuitive example based interface is a key step in the automation of large schema integration, which requires converting their underlying inductive bias into a sample selection heuristic.

9. CONCLUSION

We built a Version Spaces model for query discovery by example and developed the Sphinx learning algorithm, by adding a new kind of label and a new learning rule to the two labels and the two rules in the original Version Spaces algorithm. This new algorithm allows full and accurate verification by a user of potential mappings of semantic relationships. This is accomplished entirely by example, where only the initial data-mapping example needs to be supplied by the user. The active learning and sample selection system incorporated in Sphinx generate additional examples, which are labeled positive or negative by the user. We present a heuristic, motivated for data federating queries, which minimizes the number of examples submitted to the user by quickly eliminating potential features.

With Sphinx we grapple with the issue of designing a GUI Interface for complex data transformations. There already are successful semi-automated schema-matching tools. However, as those semantic tools currently require an expert to check correctness, we sought to identify a first set of advanced semantic elements which could still be brought to the level of a GUI for a non-technical user.

We note that Sphinx, has no real intelligent understanding of the semantic properties of data. Rather it focuses on a purely syntactic understanding of the federating views and on a user interaction model. Sphinx relies entirely upon the user to provide semantic knowledge and does not seek to derive it from context and from existing information. This split between user and system responsibilities was motivated by our specific goals. However, future work could incorporate machine learning techniques found in current schema-matching tools that exploit such information. Without abandoning the goal of accuracy, the system can make more educated guesses to improve current heuristics, converge quicker in complex situations and handle larger search spaces.

Future work should also address a wider range of practical issues in schema integration within the framework we established. In particular, we leave open the issue of adjusting and testing the current Sphinx strategy to handle different kinds of queries, such as aggregation operators and disjunction, and incorporating Skolem functions to generate new key values. Broadening the generality of Sphinx in this fashion will require expanding the current GUI principles to require several kinds of user input beyond our initial minimalist approach. It will

also mean drawing on the expertise of existing schema-matching techniques that handle a wide range of schematic heterogeneity.

Bibliography

- Abiteboul S., S. Cluet, T. Milo (1997): Correspondence and Translation for Heterogeneous Data. *ICDT Conference 1997*, pages: 351-363
- Baumgartner R., S. Flesca, G. Gottlob (2001): Visual Web Information Extraction with Lixto. *VLDB Conference 2001*, pages: 119-128.
- Castano S., De Antonelli V. (1999): A schema analysis and reconciliation tool environment. *IDEAS Conference 1999*: 53-62.
- Clifton C., E. Housman, A. Rosenthal (1997): Experience with a Combined Approach to Attribute-Matching Across Heterogeneous Databases. *IFIP TC2/WG2.6 Seventh Conference on Database Semantics (DS-7)*, pages: 428-452.
- Cluet S., C. Delobel, J. Siméon, K. Smaga (1998): Your Mediators Need Data Conversion ! *SIGMOD Conference 1998*: 177-188
- Cohn D., L. Atlas, R. Ladner (1994): Improving Generalization with Active Learning. *Machine Learning 15(2)*: 201-221.
- Crescenzi V., G. Mecca, P. Merialdo: RoadRunner (2001): Towards Automatic Data Extraction from Large Web Sites. *VLDB Conference 2001*: 109-118.
- Dagan I., S. Engelson (1995): Committee-Based Sampling For Training Probabilistic Classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 150-157.
- Dhamankar R., Y. Lee, A. Doan, A. Halevy, P. Domingos (2004): iMAP: Discovering Complex Mappings between Database Schemas. *SIGMOD Conference 2004*, pages: 383-394.
- Doan A., P. Domingos, A. Halevy (2001): Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. *SIGMOD Conference 2001*.
- Fernandez M., A. Morishima, D. Suci (2001): Efficient Evaluation of XML Middleware Queries. *SIGMOD Conference 2001*.
- Florescu D., A. Levy, A. Mendelzon (1998): Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record 27(3)*, pages: 59-74.
- Garcia-Molina H., Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, J. Widom (1997): The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems 8(2)*: 117-132.
- Grannis S., J. Overhage, S. Hui, C. McDonald (2003): Analysis of a Probabilistic Record Linkage Technique without Human Review. *JAMIA (Symposium Supplement) Proceedings of the American Medical Informatics Association Annual Symposium*, pages: 259-263.
- Grannis S., J. Overhage, S. Hui, C. McDonald (2002): Analysis of Identifier Performance using a Deterministic Linkage Algorithm. *JAMIA (Symposium Supplement) Proceedings of the American Medical Informatics Association Annual Symposium*, pages: 305-309.
- Haas L., D. Kossman, E. Wimmers, J. Yang (1997): Optimizing Queries Across Diverse Data Sources. *VLDB Conference 1997*: 276-285.
- Haussler D. (1988): Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artif. Intell. 36(2)*: 177-221.
- Hirsh H. (1991): Theoretical Underpinnings of Version Spaces. *IJCAI Conference 1991*, 665-670.
- Hirsh H. (1994): Generalizing Version Spaces. *Machine Learning 17(1)*: 5-46.

- Idemstam-Almquist P. (1990): Demand networks: an alternative representation of version spaces. *Master's thesis*, Department of Computer Science and Systems Sciences, The Royal Institute of Technology and Stockholm University, Stockholm, Sweden.
- Kent W. (1991): Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language. *VLDB Conference 1991*: 147-160/
- Kent W. (1992): Profile Functions and Bag Theory. *Hewlett-Packard Company*. Technology Department, Hewlett-Packard Laboratories. Palo Alto, California.
- Krishnamurthy R., W. Litwin, W. Kent (1991): Language Features for Interoperability of Databases with Schematic Discrepancies. *SIGMOD Conference 1991*: 40-49.
- Lakshmanan L., F. Sadri, I. Subramanian (1996): SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems. *VLDB Conference 1996*: 239-250.
- Lau T., S. Wolfman, P. Domingos, D. Weld (2003): Programming by Demonstration using Version Space Algebra, *Machine Learning* 53(1-2): 111-156.
- Lesh N., O. Etzioni (1996): Scaling Up Goal Recognition. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 244-255.
- Levy A., A. Rajaraman, J. Ordille (1996): Querying Heterogeneous Information Sources Using Source Descriptions. *VLDB Conference 1996*: 251-262.
- Lewis D., J. Catlett (1994): Heterogenous Uncertainty Sampling for Supervised Learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages: 148-156.
- Li W., C. Clifton, S. Liu (2000): SemInt: a tool for identifying attribute correspondences in heterogeneous databases using neural network. *Data and Knowledge Engineering* 33(1): 49-84.
- MacKay D. (1992): Information-based objective functions for active data selection, *Neural Computation* 4(4), pages: 590-604.
- Madhavan J., P. Bernstein, E. Rahm (2001): Generic Schema Matching with Cupid. *VLDB Conference 2001*, pages: 49-58.
- Miller R., L. Haas, M. Hernández (2000): Schema Mapping as Query Discovery. *VLDB Conference 2000*, pages: 77-88.
- Milo T., S. Zohar (1998): Using Schema Matching to Simplify Heterogeneous Data Translation. *VLDB Conference 1998*, pages: 122-133.
- Mitchell T. (1977): Version Spaces: A Candidate Elimination Approach to Rule Learning. *IJCAI Conference 1977*, pages: 305-310
- Mitchell T. (1978): Version Spaces: An Approach to Concept Learning. *PhD thesis*, Stanford University, December 1978. *Stanford CS report* STAN-CS-78-711, HPP79 -2.
- Mitra P., G. Wiederhold, M. Kersten (2000): A Graph-Oriented Model for Articulation of Ontology Interdependencies. *EDBT Conference 2000*, pages: 86-100.
- Muslea I., S. Minton, C. Knoblock (2000): Selective Sampling with Redundant Views. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages: 621-626.
- Palopoli L., G. Terracina, D. Ursino (2000): The System DIKE: Towards the Semi-Automatic Synthesis of Cooperative Information Systems and Data Warehouses. *ADBIS-DASFAA Conference 2000*, pages: 108-117.
- Park Y., Han Y., Choi K. (1995): Automatic Thesaurus Construction Using Bayesian Networks. *CIKM Conference 1995*, pages: 212-217.
- Popescu A., O. Etzioni, H. Kautz (2003): Towards a theory of natural language interfaces to databases. *International Conference on Intelligent User Interfaces*, pages: 149-157.
- Rahm E., P. Bernstein (2001): A survey of approaches to automatic schema matching. *VLDB Journal* 10(4), pages: 334-350.

- Scheuermann P., W.-S. Li, C. Clifton: Multidatabase Query Processing with Uncertainty in Global Keys and Attribute Values. *Journal of the American Society for Information Science* 49(3), pages: 283-301.
- Seung H., M. Opper, H. Sompolinsky (1992): Query by Committee. In *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory*, pages: 287-294.
- Smirnov E. (2001): Conjunctive and disjunctive version spaces with instance-based boundary sets. *PhD thesis*, Dept. of Computer Science, Maastricht University, Maastricht, The Netherlands.
- Sheth A., J. Larson (1990): Federated Database Systems for managing Distributed Heterogeneous and Autonomous Databases. *ACM Computing Surveys* 22(3), pages: 183-236.
- Takenobu T., I. Makoto, T. Hozumi (1995): Automatic Thesaurus Construction Based on Grammatical Relations. *IJCAI Conference 1995*, pages:1308-1313.
- Thompson C., M. Califf, R. Mooney (1999): Active Learning for Natural Language Parsing and Information Extraction. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages: 406-414.
- Tomasic A., L. Raschid, P. Valduriez (1996): Scaling Heterogeneous Databases and the Design of Disco. *ICDCS Conference 1996*, pages: 449-457.
- Vassalos V., Y. Papakonstantinou (1997): Describing and Using Query Capabilities of Heterogeneous Sources. *VLDB Conference 1997*, pages: 256-265.
- Vidal M., L. Raschid, J. Gruser (1998): A Meta-Wrapper for Scaling up to Multiple Autonomous Distributed Information Sources. *CoopIS 1998*, pages: 148-157
- XML Query: <http://www.w3.org/TR/XQuery>
- Yan L., M. Özsu, L. Liu (1997): Accessing Heterogeneous Data Through Homogenization and Integration Mediators. *CoopIS 1997*, pages: 130-139.
- Yan L., R. Miller, L. Haas, R. Fagin (2001): Data Driven Understanding and Refinement of Schema Mappings. *SIGMOD Conference 2001*.
- Zloof M. (1977): Query-by-Example: A Data Base Language. *IBM Systems Journal*.

Appendix

Formal proofs are included in this Appendix for the referees' consideration.

Lemma 3:

Let (fv, pos) be a labeled pair, q a query in $QS(s, G)$ and $QS' = R_p(fv)(QS)$:
 $q \notin QS' \Leftrightarrow (\forall dm : FV(dm) = fv \Rightarrow dm \notin DM^+(q))$

Proof:

Assume $fv = (e_1, e_2, \dots, e_{pf})$, $q = (q_1, q_2, \dots, q_{pf})$
 $QS = (s, G)$, $QS' = (s', G')$
 $G = \{ (g_{1,1}, \dots, g_{1,pf}),$
 $(g_{2,1}, \dots, g_{2,pf}),$
 \dots
 $(g_{k,1}, \dots, g_{k,pf}) \}$

- $q \notin QS' \Rightarrow (\forall dm : FV(dm) = fv \Rightarrow dm \notin DM^+(q))$

Assume dm such that $FV(dm) = (e_1, e_2, \dots, e_{pf})$.

$q \in QS, q \notin QS' \Leftrightarrow (\forall i: q_i \leq s_i) \wedge (\exists i: g_i \leq q) \wedge ((\exists i: q_i > s_i') \vee (\forall i: \neg(g_i' \leq q)))$

and since $\forall i: g_i = g_i'$:

$$\Leftrightarrow (\forall i: q_i \leq s_i) \wedge (\exists i: q_i > s_i') \wedge (\exists i: g_i \leq q)$$

$$\Leftrightarrow (\exists i: s_i \geq q_i > s_i') \wedge (q \in QS)$$

and since $s_i > s_i'$, we must have $s_i = 1$ and $s_i' = 0$, which by definition of s' and R_p implies $e_i=0$:

$$\Rightarrow (\exists i: q_i > e_i)$$

$$\Rightarrow dm \notin DM^+(q)$$

- $(\forall dm : FV(dm) = fv \Rightarrow dm \notin DM^+(q)) \Rightarrow q \notin QS'$

Assume dm such that $FV(dm) = (e_1, e_2, \dots, e_{pf})$.

$dm \notin DM^+(q) \Rightarrow (\exists i: q_i > e_i)$

$$\Leftrightarrow (\exists i: q_i = 1 \wedge e_i = 0)$$

and since $q \in QS$, we have $(\forall i: q_i \leq s_i) \wedge (\exists i: g_i \leq q)$

$$\Rightarrow (\exists i: q_i = 1 \wedge e_i = 0 \wedge q_i \leq s_i)$$

$$\Rightarrow (\exists i: q_i = 1 \wedge e_i = 0 \wedge s_i = 0)$$

and since by definition of s' and R_p , s_i' must be 0 in that case :

$$\Rightarrow (\exists i: q_i = 1 \wedge s_i = 1 \wedge s_i' = 0)$$

$$\Rightarrow (\exists i: q_i > s_i')$$

$$\Rightarrow q \notin QS'$$

■

Lemma 4:

Let (fv, pos) be a labeled pair, q a query in $QS(s, G)$ and $QS' = R_n(fv)(QS)$:
 $q \notin QS' \Leftrightarrow (\forall dm : FV(dm) = fv \Rightarrow dm \in DM^+(q))$

Proof:

Assume $fv = (e_1, e_2, \dots, e_{pf})$, $q = (q_1, q_2, \dots, q_{pf})$

$QS = (s, G)$, $QS' = (s'', G'')$

$s'' = s$

$G = \{(g_{1,1}, \dots, g_{1,pf}),$

$(g_{2,1}, \dots, g_{2,pf}),$

\dots

$(g_{k,1}, \dots, g_{k,pf})\}$

$G'' = \{ (g_{i,1}', \dots, g_{i,pf}') \mid [(g_{i,1}', \dots, g_{i,pf}') \leq s'] \wedge$

$[[(\exists p: (g_{p,1}, \dots, g_{p,pf}) \in G) \wedge (\forall j: e_j=0 \Rightarrow g_{p,j}=0) \wedge (\exists f: (\forall j \neq f: g_{i,j}' = g_{p,j}) \wedge e_f=0 \wedge g_{i,f}' = 1)]$

$\vee [(\exists p: (g_{p,1}, \dots, g_{p,pf}) \in G) \wedge (\exists f: e_f=0 \wedge g_{p,f}=1) \wedge (\forall j: g_{p,j} = g_{i,j}')]] \}$

- $q \notin QS' \Rightarrow (\forall dm : FV(dm) = fv \Rightarrow dm \in DM^+(q))$

Assume dm such that $FV(dm) = (e_1, e_2, \dots, e_{pf})$

since $s'' = s : q \notin QS' \Rightarrow (\exists j \forall i: q_i \geq g_{j,i}) \wedge (\forall j \exists i: q_i < g_{j,i})$.

Let z be such that $\forall i: q_i \geq g_{z,i}$

Assume A: $(\exists i: e_i = 0 \wedge g_{z,i} = 1)$

in that case $g_z \in G''$ (by fulfilling the second part of the disjunction),

and because $g_z \leq q \leq s'' = s$, we find that $q \in QS'$ which is impossible.

We therefore deduce $\neg A: (\forall i: (e_i = 0) \Rightarrow (g_{z,i} = 0))$

Let f be any f such that $e_f = 0$, we define the vector $ng = (ng_1, ng_2, \dots, ng_{pf})$

with $ng_f = 1$, and $(\forall i \neq f: ng_i = g_{z,i})$

Assume B: $s_f = 0$,

then because $q \leq s$, we have $q_f = 0$

Assume $\neg B: s_f = 1$,

then by definition $ng \in G'$

and since $(\forall i \neq f: ng_i = g_{z,i})$, we have: $(\forall i \neq f: ng_i \leq q_i)$

Assume C: $q_f = 1$,

then $ng_f \leq q_f$,

and since $(\forall i: ng_i \leq q_i)$, therefore $ng \leq q \leq s'' = s$

this implies $q \in QS'$ which is impossible.

We can therefore deduce $\neg C: q_f = 0$.

QED (we have just proved $\forall f: e_f = 0 \Rightarrow q_f = 0$)

- $(\forall dm : FV(dm) = fv \Rightarrow dm \in DM^+(q)) \Rightarrow q \notin QS'$

Assume dm such that $FV(dm) = (e_1, e_2, \dots, e_{pf})$.

Let $g'' \in G'': g'' = (g_{z,1}', g_{z,2}', \dots, g_{z,pf}')$ with all the properties listed above for a member of G'' ,

in particular $(\exists p: g_p = (g_{p,1}, \dots, g_{p,pf}) \in G)$ such that:

$[(\forall j: e_j=0 \Rightarrow g_{p,j}=0) \wedge (\exists f: (\forall j \neq f: g_{i,j}' = g_{p,j}) \wedge e_f=0 \wedge g_{i,f}' = 1)]$

$\vee [(\exists f: e_f=0 \wedge g_{p,f}=1) \wedge (\forall j: g_{p,j} = g_{i,j}')]$

Take such a p :

Assume A: $(q \geq g_p)$

and since $dm \in PDM(q)$, therefore we have $(\forall i: e_i \geq q_i \geq g_{p,i})$.

The property $(\exists f: e_f = 0 \wedge g_{p,f} = 1)$ is now impossible,

therefore by definition of $G'': (\exists f: (e_f = 0) \wedge (g_{z,f}' = 1) \wedge (\forall i \neq f: g_{z,i}' = g_{p,i}'))$.

Take such an f :

because $(e_f = 0)$ and $(\forall i: e_i \geq q_i)$

the only possibility is in that case: $q_f = e_f = 0$
 $(q_f = 0)$ and $(g_{z,f}' = 1)$ imply $\neg(g'' \leq q)$

Assume $\neg A$: $(\exists i: q_i < g_{p,i})$

Assume B: $(\forall j: g_{z,j}' = g_{p,i})$

then $(\exists i: q_i < g_{p,i} = g_{z,i}')$ which implies $\neg(g'' \leq q)$

Assume $\neg B$: $\neg(\forall j: g_{z,j}' = g_{p,i})$

Then by definition of G'' , the other part of the disjunction must be true:

$[(\forall j: e_j=0 \Rightarrow g_{p,j}=0) \wedge (\exists f: (\forall j \neq f: g_{z,j}' = g_{p,j}) \wedge e_f=0 \wedge g_{z,f}' = 1)]$

in particular: $(\exists f: g_{z,f}' = 1 \wedge (\forall j \neq f: g_{z,j}' = g_{p,j}))$

therefore: $(\forall j: g_{z,j}' \geq g_{p,j})$.

Recall that $(\exists i: q_i < g_{p,i}) \Rightarrow (\exists i: q_i < g_{p,i} \leq g_{z,i}') \Rightarrow \neg(g'' \leq q)$

QED (we have just proved $(\forall g'' \in G'': \neg(g'' \leq q))$)

■

Lemma 5(k):

If:

$LS_k = ((fv_1, l_1), \dots, (fv_k, l_k))$ is a label sequence,

and RS_k is a rule sequence triggered by LS_k such that $(R_a(p_1), R_a(p_2), \dots, R_a(p_a))$ is the exact subsequence of applications of the *additional rule* in RS_k ,

and $VS(RS_k) = QS(s_k, G_k)$,

and q is compatible with LS_k ,

and q' a query such that $(\forall j \leq a: q_{p_j}' = 1) \wedge (\forall i: (\forall j \leq a: i \neq p_j) \Rightarrow (q_i' = q_i))$

Then:

$q' \in VS(RS_k)$

Proof:

Lemma 5 is parameterized by k , the length of the label sequence. The proof is an induction on k .

- $k = 0$

$LS_0 = \emptyset, RS_0 = \emptyset, a=0$

We simply verify that $q = q'$ and that both are in $VS(\emptyset) = QS(s_0, G_0)$ which is the whole search space.

- Assume Lemma 5(k) is true: prove Lemma 5(k+1)

Case 1:

The subsequence of applications of *additional rule operators* is the same for RS_{k+1} and RS_k . In other terms there is no application of the *additional rule operator* between step k and step $k+1$.

Assume q and q' are queries such that q is compatible with LS_{k+1} and q' is such that $(\forall j \leq a: q_{p_j}' = 1) \wedge (\forall i: (\forall j \leq a: i \neq p_j) \Rightarrow (q_i' = q_i))$.

We can apply the induction hypothesis, Lemma 5(k) on LS_k, RS_k, q and q' : therefore $q' \in VS(RS_k)$.

Assume that $fv = (e_1, e_2, \dots, e_{pf})$.

There are three further cases on the value of the label l_{k+1} :

- If $l_{k+1} = \text{pos}$, then there exists $dm_{k+1} \in DM^+(q)$, such that $FV(dm_{k+1}) = fv_{k+1}$ because q is compatible with LS_{k+1} .

$dm_{k+1} \in PDM(q) \Rightarrow (\forall i: e_i \geq q_i)$

There are two further cases:

- Let i be such that $(\forall j \leq a: i \neq p_j)$, then $q_i' = q_i$ and $e_i \geq q_i'$
- Let i be such that $(\exists j \leq a: i = p_j)$, then
let j be such that $j \leq a$ and $i = p_j$

Assume A: ($e_i = 0$), then $fv_{k+1} \in 0_i$,
then Precondition for $R_a(i=p_j)$ in the rule sequence, dictates that the
label l_{k+1} be negative or missing. This is impossible.

We can therefore deduce $\neg A$: ($e_i = 1$), and $e_i \geq q_i'$ is assured.

Thus we proved with both cases that $(\forall i: e_i \geq q_i')$,
and therefore that $dm_{k+1} \in PDM(q')$.

Using Lemma 3, we can deduce that $q' \in R_p(fv_{k+1})(s_k, G_k)$,
therefore $q' \in VS(RS_{k+1})$.

- If $l_{k+1} = \text{neg}$, then there exists $dm_{k+1} \notin PDM(q)$ such that $FV(dm_{k+1}) = fv_{k+1}$, because q is compatible with LS_{k+1}

$dm_{k+1} \notin DM^+(q) \Rightarrow (\exists i: e_i < q_i)$.

Let i be such that $e_i < q_i$. There are two further cases:

- o $(\forall j \leq a: i \neq p_j)$, then $q_i = q_i'$ and $e_i < q_i'$. Therefore $dm_{k+1} \notin PDM(q')$
- o $(\exists j \leq a: i = p_j)$, then let j be such that $j \leq a$ and $i = p_j$.
 $e_i < q_i \Rightarrow e_i = 0$, and since $q_i' = 1$, $e_i < q_i$ is assured. Therefore $dm_{k+1} \notin PDM(q')$.

With both cases we established $dm_{k+1} \notin PDM(q')$.

Using Lemma 4, we can deduce that $q' \in R_n(fv_{k+1})(s_k, G_k)$,

and since we are in the case where there is no application of the *additional rule*
between step k and step $k+1$: $q' \in VS(RS_{k+1})$

- If $l_{k+1} = \text{mis}$, then since there is no application of the *additional rule*, $RS_k = RS_{k+1}$
and $q' \in VS(RS_{k+1})$

Case 2:

The subsequence of applications of the *additional rule operator* is incremented from RS_k to RS_{k+1}
by the application of $R_a(p_{a+1})$. In other terms $R_a(p_{a+1})$ is applied between step k and step $k+1$.

Assume q and q' are queries such that q is compatible with LS_{k+1} and q' is such that $(\forall j \leq a+1$:
 $q_{p_j}' = 1) \wedge (\forall i: (\forall j \leq a+1: i \neq p_j) \Rightarrow (q_i' = q_i))$.

Let $q'' = (q_1'', q_2'', \dots, q_{p_f}'')$ be such that $(\forall j \leq a: q_{p_j}'' = 1) \wedge (\forall i: ((\forall j \leq a: i \neq p_j) \Rightarrow q_i'' = q_i))$.

We can apply the inductive hypothesis, Lemma 5(k) to q'' : $q'' \in VS(RS_k)$.

Assume that $fv_{k+1} = (e_1, e_2, \dots, e_{p_f})$.

There are two further cases:

- If $l_{k+1} = \text{neg}$, then there exists $dm_{k+1} \notin PDM(q)$ such that $FV(dm_{k+1}) = fv_{k+1}$, because q is compatible with LS_{k+1}

$dm_{k+1} \notin DM^+(q) \Rightarrow (\exists i: e_i < q_i)$

There are two further cases:

- o Let i be such that $(\forall j \leq a: i \neq p_j)$, then $q_i'' = q_i$ and therefore $e_i < q_i''$
- o Let i be such that $(\exists j \leq a: i = p_j)$, then
let $j \leq a$ such that $i = p_j$: in that case $q_i'' = 1$
and since $e_i < q_i$. The only possibility is $e_i = 0 < q_i'' = 1$.

Thus we proved with both cases that $(\exists i: e_i < q_i'')$

$\Rightarrow dm_{k+1} \notin PDM(q'')$.

Using Lemma 4: $q'' \in R_n(fv_{k+1})(s_k, G_k)$.

Note that $(\forall i \neq p_{a+1}: q_i'' = q_i') \wedge (q_{p_{a+1}}' = 1)$.

There are two cases:

- o Case A: $q_{p_{a+1}} = 1$: in this case since $q_{p_{a+1}}'' = q_{p_{a+1}}$, we have $q'' = q'$,
and since $q_{p_{a+1}}' = 1$, by definition of the action for $R_a(p_{a+1})$:

$q'' \in R_a(p_{a+1})(R_n(fv_{k+1})(s_k, G_k))$

Therefore $q' \in VS(RS_{k+1})$.

○ Case B: $q_{pa+1} = 0 : q_{pa+1}'' = 0, q_{pa+1}' = 1$

We will prove that q' is compatible with LS_{k+1} , then we will deduce that $q' \in VS(RS_{k+1})$.

- Let $i \leq k+1$ be such that $(fv_i, l_i = \text{pos}) \in LS_k$.
There exists $dm_i \in PDM(q'')$ s.t. $FV(dm_i) = fv_i = (f_1, \dots, f_{pf})$
 $dm_i \in VM(q'') \Rightarrow (\forall i: f_i \geq q_i'')$.
If $f_{pa+1} = 1$ then $(\forall i: f_i \geq q_i') \Rightarrow dm_i \in PDM(q')$
If $f_{pa+1} = 0$ then by the precondition for $R_a(p_{a+1})$, l_i is neg or mis, which is impossible.
Therefore $dm_i \in PDM(q')$.
- Let $i \leq k+1$ be such that $(fv_i, l_i = \text{neg}) \in LS_k$
There exists $dm_i \notin DM^+(q'')$ s.t. $FV(dm_i) = fv_i = (f_1, \dots, f_{pf})$
In this case $dm_i \notin PDM(q'')$, which implies that $\exists i: f_i < q_i''$.
If $i = pa+1$ then since $q_{pa+1}'' = 0$, there is a contradiction.
If $i \neq pa+1$. In that case: $q_i'' = q_i'$, which implies that $f_i < q_i'' = q_i'$.
Therefore $\exists i: f_i < q_i'$ and $dm_i \notin DM^+(q')$.

We have just established that q' is compatible with LS_{k+1} .

By induction on the label sequence LS_k , we now prove that $q' \in VS(RS_{k+1})$:

- $q' \in VS(\emptyset)$. q' is in the initial search space.
 - assume $q' \in VS(RS_i) = QS(s_i, G_i)$, RS_i triggered by LS_i , $LS_{i+1} = LS_i + (fv_i, l_i)$
 - If $l_i = \text{pos}$, since q' is compatible with LS' , there exists $dm_i \in DM^+(q')$ s.t. $FV(dm_i) = fv_i$ and by Lemma 3, $q' \in VS(RS_i + R_p(fv_i))$
 - If $l_i = \text{neg}$, since q' compatible with LS' , there exists $dm_i \notin PDM(q')$ s.t. $FV(dm_i) = fv_i$ by Lemma 4, $q' \in R_n(fv_i)(s_i, G_i)$.
 $\forall j \leq a+1: q_{pj}' = 1$ and by definition of the $R_a(p_j)$: $(\forall j: q' \in R_a(p_j)(R_n(fv_i)(s_i, G_i)))$.
Since by definition of the algorithm, when $l_i = \text{neg}$, RS_{i+1} is equal to either $RS_i + R_n(fv_i)$ or $RS_i + R_n(fv_i) + R_a(p_j)$ for some j :
 $q' \in VS(RS_{i+1})$.
 - If $l_i = \text{mis}$, then
 $\forall j \leq a+1, q_{pj}' = 1$, and by definition of the $R_a(p_j)$ operator, we have $\forall j: q' \in R_a(p_j)(s_i, G_i)$.
Since by definition of algorithm, RS_{i+1} is equal to either RS_i or $RS_i + R_a(p_j)$ for some j :
 $q' \in VS(RS_{i+1})$.
- If $l_{k+1} = \text{mis}$, there are two cases:
- If $q_{pa+1} = 1$, then $q' = q''$.
 $q' = q'' \in VS(RS_k)$,
and since $q_{pa+1}' = 1$, by definition of $R_a(p_{a+1})$:
 $q' \in R_a(p_{a+1})(s_k, G_k) = VS(RS_{k+1})$.
 - If $q_{pa+1}' = 0$. Same scenario and same proof as in Case B above. We first prove that q' is compatible with LS_{k+1} , then by a mini-induction that $q' \in VS(RS_{k+1})$. QED

■

Theorem 2:

Let LS be a label sequence such that LS contains 2^{pf} distinct, correctly labeled pairs, one for every possible feature vector instance. Let RS be the rule sequence triggered by LS, then $VS(RS) = QS(s, G)$ has converged to a single query.

Proof:

In order to prove that $VS(RS)$ has converged to a single query, we will prove that partial convergence has been reached for each of the potential features.

Let $f \leq pf$, be a potential feature:

- Assume A: $[\forall i: (((dm_i = (e_1, e_2, \dots, e_{pf}), l_i) \in LS) \wedge (e_f = 0)) \Rightarrow l_i = \text{neg or mis}]$
Since LS contains all possible feature vectors, the precondition for $R_a(f)$ is fulfilled.
Therefore $R_a(f)$ is guaranteed to be in the rule sequence RS and by definition of $R_a(f)$:
 $(\forall i: s_f = g_{i,f} = 1)$.
The algorithm has partially converged on feature f.
- Assume $\neg A$: $[\exists i: (((dm_i = (e_1, e_2, \dots, e_{pf}), l_i) \in LS) \wedge (e_f = 0)) \Rightarrow l_i = \text{pos}]$
In that case there exists i such that $R_p(FV(dm_i))$ is in the rule sequence RS.
Since $e_f = 0$, by definition of $R_p(FV(dm_i))$: $(\forall i: s_f = g_{i,f} = 0)$.
The algorithm has partially converged on feature f.

Thus the algorithm has converged on all potential features, and $(\forall i: s_f = g_{i,f})$, and using Theorem 1 has converged to a single query.

■