Automating Schema Mapping for Data Integration

Li Xu* and David W. Embley*
Department of Computer Science
Brigham Young University
Provo, Utah 84602, U.S.A.
{lx, embley}@cs.byu.edu

Abstract

To integrate data from disparate, heterogeneous information sources in an open environment, data-integration systems demand a resolution to specify mappings between target and source schemas. Automating schema mapping is challenging. Previous approaches (e.g. [MBR01, DDH01]) to automating schema mapping focus on computing direct matches between two schemas. Schemas, however, rarely match directly. Thus, to complete the task of schema mapping, we must also compute indirect matches. In this paper, we present a framework for generating a source-to-target mapping containing direct as well as many indirect matches between a source schema and a target schema. Recognizing expected data values associated with schema elements and applying schema-structure heuristics are the key ideas to computing indirect matches. Experiments we have conducted over several real-world application domains show encouraging results, yielding over 90% precision and recall measures for both direct and indirect matches. We formalize direct and indirect matches as mapping elements in source-to-target mappings by slightly extending the relational algebra. Based on source-to-target mappings, we offer a unified approach for data integration. Compared with other data integration approaches, our data-integration approach combines their advantages, mitigates their disadvantages, and provides a viable alternative for flexible and scalable data integration.

1 Introduction

Data integration refers to the problem of combining data residing at autonomous and heterogeneous sources and providing users with a unified global schema [Ull97, Hal01, CCGL02]. Two main concepts constitute the architecture of a data-integration system [Ull97]: wrappers and mediators. A *wrapper* wraps an information source and models the source using a *source schema*. A *mediator* maintains a *global schema* and *mappings* between the global and source schemas. We focus here on data-integration systems that do not materialize data in the global schema. Currently, there are two main initiatives to integrate data and answer queries without materializing a global schema: Global-as-view (GAV) [CGMH+94] and Local-as-View(LAV) [LRO96, GKD97]. In either a GAV or LAV approach, whenever a user poses a query in terms of relations in the global schema, the mediator within the data-integration system uses a *query-reformulation* procedure to translate the

*This material is based upon work supported by the National Science Foundation under grant IIS-0083127.

query into sub-queries that can be executed in sources such that the mediator can collect returned answers from the sources and combine them as the answer to the query.

One of the important applications for data integration is to deal with the explosion of data on the World Wide Web. E-business applications such as comparison shopping and knowledge-gathering applications such as vacation planning raise the following major issues for approaches to data integration. (1) *Heterogeneity.* The sources are autonomous, establishing their own vocabulary and structure. (2) *Scalability.* The number of sources to access and integrate is large. (3) *Continual infusion and change of local information sources.* New sources continually become available and become part of the system. Sources within the system may change frequently. (4) *Query process complexity.* Users frequently pose queries, which can be very complex, with respect to the collection of information sources. (5) *Evolution.* As applications evolve, Database Administrators (DBAs) may wish to change the global schema to include some new items of interest. Most of the existing approaches to data integration [CGMH+94, LRO96, GKD97, Ull97, FLM99, MHH+01, CCGL02], however, have addressed only some of these issues. Thus, they do not meet the needs of the modern E-business applications.[1] To address these issues, we present an alternative point of view, called TIQS (Target-based Integration Query System).

TIQS largely depends on schema mapping to resolve the five major issues in a unified framework. Schema mapping is a challenging problem. It takes two schemas as input and produces semantic correspondences between schema elements in the two input schemas [RB01]. In this paper, we assume that we wish to map schema elements from a *source* schema into a *target* schema, where the target schema corresponds to a global schema in data-integration systems. We express each semantic correspondence as a *mapping element*, which binds two schema elements if the two schema elements are semantically equivalent. In its simplest form, a mapping element is a *direct match* that binds a source schema element directly to a target schema element. To date, most research [BCV99, DDH01, EJX01, LC00, MBR01, MZ98, PTU00] has focused on computing direct matches. Such simplicity, however, is rarely sufficient, and researchers have thus proposed the use of queries over source schemas to bind with target schema elements [BE03, MHH00]. In this more complicated form, a mapping element is an *indirect match* that binds a view over the source schema, which is

[1] After describing our proposed solution we explain in Section 7 by way of comparison, how other proposed solutions fail to address one or more of these issues.

a *virtual* source schema element, to a target schema element through appropriate queries over a source schema.

We assume that all source and target schemas are described using rooted conceptual-model graphs (a conceptual generalization of XML). We augment schemas with a variety of ontological information. For this paper the augmentations we discuss are WordNet [Mil95], sample data, and regular-expression recognizers. For each application, we construct a lightweight domain ontology [ECJ+99], which declares the regular-expression recognizers. Based on the graph structure and these augmentations, we exploit a broad set of techniques together to settle direct and indirect matches between a source schema and a target schema. As will be seen, regular-expression recognition over sample data and applying schema structural characteristics are the key ways to detect indirect matches. Based on the discovered matches, the solution proposed here maps a source schema into a target schema through a source-to-target mapping, which is formally specified such that data in the source is readily accessible to load into the target.

By applying source-to-target mapping based on a predefined target schema, TIQS provides a unified, scalable approach to data integration , which combines the advantages and avoids the limitations of both GAV and LAV. TIQS has polynomial-time query reformulation, and is easy to add or modify information sources. Moreover, DBAs create the target schema and wrap source schemas independently, so that neither the target schema nor the source schemas are contingent respectively on the source schemas or the target schema. Even when DBAs modify or add new items of interest to the target schema, TIQS applies the schema mapping techniques to semi-automatically generate or adjust required source-to-target mappings between source schemas and the new target schema. Thus, TIQS increases both scalability and usability as compared to previously proposed data-integration approaches.

In this paper, we offer the following contributions: (1) a way to discover many indirect semantic correspondences between a source schema S and a target schema T as well as the direct correspondences, (2) an extension of relational algebra to express source-to-target mappings, (3) a unified approach to data integration applying source-to-target mapping, and (3) experimental results of our implementation to show that our solution performs as well (indeed better) than other approaches for direct matches and also performs exceptional well for the indirect matches with which we work. We present the details of our contribution as follows. Section 2 explains the internal representation

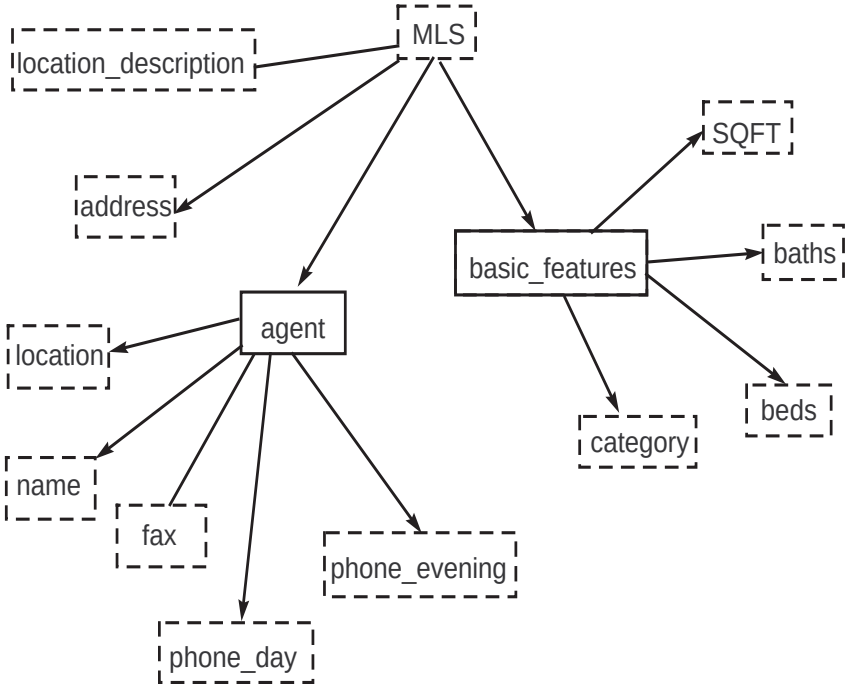
of input target and source schemas and output mappings for schema mapping. Section 3 describes a set of basic matching techniques to find potential mapping elements between elements in S and elements in T, and to provide confidence measures between 0 (lowest confidence) and 1 (highest confidence) for each potential match. Section 4 presents an algorithm to settle direct and indirect matches in a source-to-target mapping between S and T. In Section 5 we describe TIQS that mitigates the disadvantages and combines the advantages of basic data-integration approaches. Section 6 gives experimental results for a data set used in [DDH01] to demonstrate the success of our approach. In Section 7 we review related work, and in Section 8 we summarize, consider future work, and draw conclusions.

2 Internal Representation for Schema Mapping

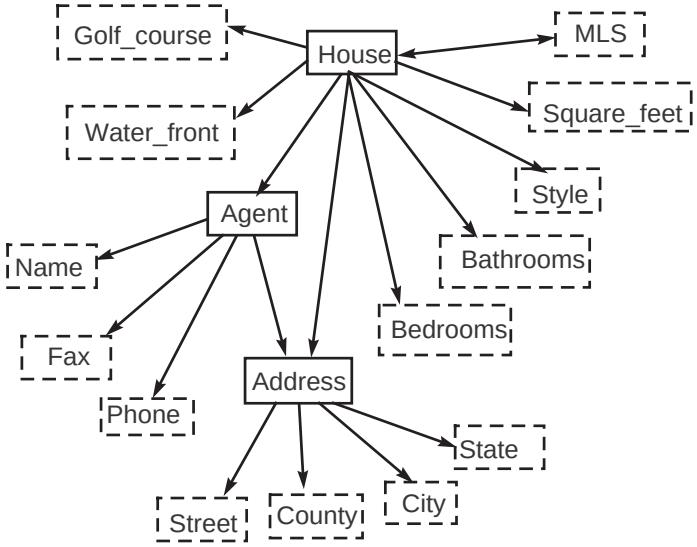
In a data integration system, a mediator maintains a target schemas, and a wrapper wraps a particular information source as a source schema. It is possible that the wrappers uses their own expression languages to describe the source schema in insolation. In this paper, we assume that we use a model language OSM-L [EKW92] to describe both source and target schemas, and an extension of the relational algebra to describe source-to-target mappings.

2.1 Target and Source Schemas

We use rooted graphs to represent both the target schema and the source schemas as conceptual specifications. Each of conceptual schemas consists of two components. One contained component is an *object/relationship-model instance* that describes sets of objects, sets of relationships among objects, and constraints over object and relationship sets. In each conceptual schema H, we denote O_H as a set of object sets and R_H as a set of relationship sets in H. An object set has associated either data values or object identifiers, which we respectively call *lexical object sets* or *non-lexical object sets*. A relationship set has associated values to represent relationships connecting object sets. The root node is a designated object of primary interest. Figure 1, for example, shows two schema structures. In a schema structure we denote lexical object sets as dotted boxes, non-lexical object sets as solid boxes, functional relationship sets as lines with an arrow from domain object set to range object set, and nonfunctional relationship sets as lines without arrowheads. The other component contained in a conceptual schema is a set of *data frames*, each of which defines the



(a) Schema 1



(b) Schema 2

Figure 1: Source Graphs for Schema 1 and Schema 2

```
Water_front matches [10] case insensitive
 keyword "\bwater(\s|_)?front\b";
end;

Golf_ourse matches [10] case insensitive
 keyword "\bgolf(\s|_)course\b";
end;

location_description matches [25] case insensitive
 constant
  { extract "\water(\s)?front\b"; },
  { extract "\bcorner\b"; },
  { extract "\bcul\-de\-sac\b"; },
  { extract "\blake\sfront\b"; },
  { extract "\bgolf(\s|_)course\b"; },
  { extract "\bfenced(\s)lot\b"; },
  { extract "\bview(s)?\b"; },
  { extract "\bbay\sfront\b"; },
  { extract "\bocean\sfront\b"; },
  ...
  { extract "\bbay\sfront\b"; };
end;

<DataSample>
 <OSM domain="RealEstate" application="1" nrOfFiles="2367" nrOfValues="494" />
 <ObjectSet id="10" name="location_description" />
 <Row text="Ocean front, Water front" />
 <Row text="Ocean front, Bay front, Corner, Cul-de-sac" />
 <Row text="Canal front, Water front" />
 <Row text="Bay front, Water front" />
 <Row text="Ocean front, Water front" />
 <Row text="Intracoastal, Water front" />
 <Row text="Cul-de-sac" />
 <Row text="Ocean front, Water front" />
 <Row text="Ocean front, Water front" />
 <Row text="Corner" />
 <Row text="Ocean front, Water front" />
 <Row text="Intracoastal, Ocean front, Water front" />
 <Row text="Cul-de-sac" />
 <Row text="Cul-de-sac" />
 <Row text="Canal front, Water front" />
 <Row text="Pond front, Cul-de-sac" />
 <Row text="Intracoastal, Water front" />
 <Row text="Pond front, Cul-de-sac" />
 <Row text="Cul-de-sac" />
 <Row text="Lake front" />
 <Row text="Lake front" />
 <Row text="Ocean front, Water front" />
 ...
```

Figure 2: Example of Data Frames: Lexical Appearances and Data Instances

potential contents of a lexical object set. A data frame for an object set either defines the lexical appearance of constant objects for the object set and establishes appropriate keywords that are likely to appear in a document when objects in the object set are mentioned, or lists of data instances for objects. Figure 2 shows lexical appearances for lexical object sets $Water\_front$, $Golf\_course$, and $location\_description$. A subset of data instances for lexical object set $location\_description$ is also included. When both lexical appearance and data instances are available in an object set in applications, a data frame for this object set provides both specifications of lexical appearance and a set of lexical values.

For any schema H, which is either a source schema or a target schema, we let Σ_H denote the union of O_H and R_H in H. Furthermore, our solution allows a variety of source derived data,

including missing generalizations and specializations, merged and split values, and transformation of attributes with Boolean indicators into values, and schema paths as relationships. Therefore, our solution "extends" the source schema elements in Σ_H, each of which we call *real* object or relationship set, to include views, each of which we call *virtual* object or relationship set. We let V_H denote the extension of Σ_H with derived virtual object and relationship sets.

2.2 Source-to-Target Mappings

We consider a source-to-target mapping M_{ST} between a source schema S and a target schema T as a function f_{ST}. The domain of f_{ST} is V_S. And the range of f_{ST} is Σ_T. Thus, we can denote a source-to-target mapping M_{ST} as a function $f_{ST}(V_S) \rightarrow \Sigma_T$. Intuitively, a source-to-target mapping M_i represents inter-schema correspondences between a source schema S_i and a target schema T. If we let Schema 1 in Figure 1(a) be the target and let Schema 2 in Figure 1(b) be the source, for example, a source-to-target mapping between the two schemas includes a semantic correspondence, which declares that the lexical object set *Bedrooms* in the source semantically corresponds to the lexical object set *beds* in the target. If we let Schema 1 be the source and Schema 2 be the target, a source-to-target mapping declares that the union of the two sets of values in *phone_day* and *phone_evening* in the source corresponds to the values for *Phone* in the target.

We represent semantic correspondences between a source schema S and a target schema T as a set of mapping elements. A mapping element is either a *direct match* which binds a schema element in Σ_S to a schema element in Σ_T, or an *indirect match* which binds a virtual schema element in V_S to a target schema element in Σ_T through an appropriate *mapping expression* over Σ_S. A mapping expression specifies how to derive a virtual schema element through manipulation operations over a source schema. We denote a mapping element as $(t \sim s \Leftarrow \theta_s(\Sigma_S))$, where $\theta_s(\Sigma_S)$ is a mapping expression that derives a source element s in V_S,[2] and t is a target schema element in Σ_T.

2.3 The Algebra for Source-to-Target Mappings

Each object and relationship set (including derived object and relationship sets) in the target and source schemas are single-attribute or multiple-attribute relations. Thus, relational algebra directly applies to the object and relationship sets in a source or target schema. The standard

[2]Note that the mapping expression may be degenerate so that $(t \sim s)$ is possible.

operations, however, are not enough to capture the operations required to express all the needed source-to-target mappings. Thus, we extend the relational algebra.

To motivate our use of standard and extended operators, we list the following problems we must face in creating derived object and relationship sets over source schemas.

- *Union* and *Selection.* The object sets, *phone_day* and *phone_evening* in Schema 1 of Figure 1(a) are both subsets of *Phone* values in Schema 2 of Figure 1(b), and the relationship sets $agent - phone\_day$ and $agent - phone\_evening$ in Schema 1 are both specializations of $Agent - Phone$ value pairs in Schema 2. Thus, if Schema 2 is the target, we need the union of the values in *phone_day* and *phone_evening* and the union of the relationships in $agent - phone\_day$ and $agent - phone\_evening$ in Schema 1; and if Schema 1 is the target, we should use *Selection* to find a way to separate the day phones from the evening phones and separate the relationships between agents and day phones from those between agents and evening phones.

- *Merged* and *Split Values.* The object sets, *Street*, *City*, and *State* are separate in Schema 2 and merged as *address* of *house* or *location* of *agent* in Schema 1. Thus, we need to split the values if Schema 2 is the target and merge the values if Schema 1 is the target.

- *Object-Set Name as Value.* In Schema 2 the features $Water\_front$ and $Golf\_course$ are object-set names rather than values. The Boolean values "Yes" and "No" associated with them are not the values but indicate whether the values $Water\_front$ and $Golf\_course$ should be included as description values for *location_description* of *house* in Schema 1. Thus, we need to distribute the object-set names as values for *location_description* if Schema 1 is the target and make Boolean values for $Water\_front$ and $Golf\_course$ based on the values for *location_description* if Schema 2 is the target.

- *Path as Relationship Set.* The path $MLS - basic\_features - beds$ in Schema 1 semantically corresponds to the path $MLS - House - Bedrooms$ in Schema 2. Thus, if Schema 1 is the target, we need derive two virtual relationship set corresponding with the target relationship sets $MLS - basic_features$ and $basic\_features - beds$; otherwise, if Schema 2 is the target, we need derive two virtual relationship sets corresponding with the target relationship sets

$House - MLS$ and $House - Bedrooms$.

Currently, we use the following operations over source relations to resolve these problems. In the notation, a relation r has a set of attributes, which corresponds to the names of lexical or non-lexical object sets; $attr(r)$ denotes the set of attributes in r; and $|r|$ denotes the number of tuples in r. For non-standard operators, we provide examples to illustrate how to apply the operators over source relations.

- *Standard Operators. Selection* σ, *Union* $\cup$, *Natural Join* $\bowtie$, *Projection* π, and *Rename* ρ.

- *Composition* λ. The λ operator has the form $\lambda_{(A_1,...,A_n),A}r$ where each A_i, $1 \leq i \leq n$, is either an attribute of r or a string, and A is a new attribute. Applying this operation forms a new relation r', where $attr(r') = attr(r) \cup \{A\}$ and $|r'| = |r|$. The value of A for tuple t of row l in r' is the concatenation, in the order specified, of the strings among the A_i's and the string values for attributes among the A_i's for tuple t' of row l in r.

  Let r be the following relation, where $attr(r) = \{House,\ Street,\ City,\ State\}$.

House	Street	City	State
h1	339 Wymount Terrace	Provo	Utah
h2	15 S 900 E	Provo	Utah
h3	1175 Tiger Eye	Salt Lake City	Utah

  Applying the operation $\lambda_{(Street,\text{“, ”},City,\text{“, ”},Street),Address}r$ yields a new relation r', where $attr(r') = \{House,\ Street,\ City,\ State,\ Address\}$.

House	Street	City	State	Address
h1	339 Wymount Terrace	Provo	Utah	339 Wymount Terrace, Provo, Utah
h2	15 S 900 E	Provo	Utah	15 S 900 E, Provo, Utah
h3	1175 Tiger Eye	Salt Lake City	Utah	1175 Tiger Eye, Salt Lake City, Utah

- *Decomposition* γ. The γ operator has the form $\gamma^R_{A,A'}r$ where A is an attribute of r, and A' is a new attribute whose values are obtained from A values by applying a routine R. Typically, R extracts a substring from a given string to form part of a decomposition. Repeated application of γ allows us to completely decompose a string. Applying this operation forms a new relation r', where $attr(r') = attr(r) \cup \{A'\}$ and $|r'| = |r|$. The value of A' for tuple t of row l in r' is obtained by applying the routine R on the value of A for tuple t' of row l in r.

Let r be the following relation, where $attr(r) = \{House,\ Address\}$.

House	Address
h1	Provo, Utah
h2	339 Wymount Terrace, Provo, Utah
h3	1175 Tiger Eye, Salt Lake City, Utah

Applying the operation $\gamma^{R}_{Address,Street}r$, where R is a routine that obtains values of $Street$ from values of $Address$, yields a new relation r_1, where $attr(r_1) = \{House,\ Address,\ Street\}$.

House	Address	Street
h1	Provo, Utah	
h2	339 Wymount Terrace, Provo, Utah	339 Wymount Terrace
h3	1175 Tiger Eye, Salt Lake City, Utah	1175 Tiger Eye

Similarly, applying the operation $\gamma^{R'}_{Address,City}r$, where R' is a routine that obtains values of $City$ from values of $Address$, yields a new relation r_2, where $attr(r_2) = \{House,\ Address,\ City\}$.

House	Address	City
h1	Provo, Utah	Provo
h2	339 Wymount Terrace, Provo, Utah	Provo
h3	1175 Tiger Eye, Salt Lake City, Utah	Salt Lake City

- *Boolean β.* The β operator has the form $\beta^{Y,N}_{A,A'}r$, where Y and N are two constants representing Yes and No values in r, A is an attribute of r that has only Y or N values, and A' is a new attribute. The β operator requires the precondition $(attr(r) - \{A\}) \rightarrow \{A\}$. Applying this operation forms a new relation r', where $attr(r') = (attr(r) - \{A\}) \cup \{A'\}$ and $|r'| = |\sigma_{A=Y}r|$. The value of A' for tuple t in r' is the literal string A if and only if there exists a tuple t' in r such that $t'[attr(r) - \{A\}] = t[attr(r) - \{A\}]$ and $t'[A]$ is a Y value.

  Let r be the following relation, where $attr(r) = \{House,\ Water\ Front\}$.

House	Water Front
h1	Yes
h2	No
h3	Yes

  Applying the operation $\beta^{\text{“}Yes\text{”},\text{“}No\text{”}}_{Water\ Front,Lot\ Description}r$ yields a new relation r', where $attr(r') = \{House,\ Lot\ Description\}$.

House	Lot Description
h1	Water Front
h3	Water Front

- *DeBoolean* $\natural$. The $\natural$ operator has the form $\natural^{Y,N}_{A,A'}r$, where Y and N are two constants representing Yes and No values, A is an attribute of r, and A' is a new attribute. Applying this operation forms a new relation r', where $attr(r') = (attr(r) - \{A\}) \cup \{A'\}$ and $|r'| = |\pi_{attr(r)-\{A\}}r|$. The value of A' for tuple t in r' is Y if and only if there exists a tuple t' in r such that $t'[attr(r) - \{A\}] = t[attr(r) - \{A\}]$ and $t'[A]$ is the literal string A', or is N if and only if there does not exist a tuple t' in r such that $t'[attr(r) - \{A\}] = t[attr(r) - \{A\}]$ and $t'[A]$ is the literal string A'.

  Let r be the following relation, where $attr(r) = \{House,\ Lot\ Description\}$.

House	Lot Description
h1	Water Front
h1	Golf Course
h1	Mountain View
h2	Water Front
h3	Golf Course

  Applying the operation $\natural^{\text{“}Yes\text{”},\text{“}No\text{”}}_{Lot\ Description,Water\ Front}r$ yields a new relation r', where $attr(r') = \{House,\ Water\ Front\}$.

House	Water Front
h1	Yes
h2	Yes
h3	No

  Similarly, applying the operation $\natural^{\text{“}x\text{”},\text{“”}}_{Lot\ Description,Golf\ Course}r$ yields a new relation r'', where $attr(r'') = \{House,\ Golf\ Course\}$.

House	Golf Course
h1	x
h2	
h3	x

- *Skolemization* φ. The φ operator has the form $\varphi_{f_A}(r)$, where f_A is a skolem function, and A is a new attribute. Applying this operation forms a new relation r', where $attr(r') =$

$attr(r) \cup \{A\}$ and $|r'| = |r|$. The value of A for tuple t of row l in r' is a functional term that computes a value by applying the skolem function f_A over tuple t' of row l in r.
Let r be the following relation, where $attr(r) = \{House\}$.

House
h1
h2
h3

Applying the operation $\varphi_{f_{Basic\ Features}} r$ yields a new relation r', where $attr(r') = \{House,$ $Basic\ Features\}$.

House	Basic Features
h1	$f_{Basic\ Features}(h1)$
h2	$f_{Basic\ Features}(h2)$
h3	$f_{Basic\ Features}(h3)$

3 Matching Techniques

In this section we explain our four basic techniques for schema mapping: (1) terminological relationships (e.g. synonyms and hypernyms), (2) data-value characteristics (e.g. string lengths and alphanumeric ratios), (3) domain-specific, regular-expression matches (i.e. the appearance of expected strings), and (4) structure (e.g. structural similarities). For the first two techniques we obtain vectors of measures for the features of interest and then apply machine learning over these feature vectors to generate a decision rule and a measure of confidence for each generated decision. We use C4.5 [Qui93] as our decision-rule and confidence-measure generator. Moreover, even though we could apply the former three matching techniques to identify mapping elements between object or relationship sets, our solution focuses on discovering matches between object sets. The forth one applies schema structural characteristics to combine the results from the former three ones for object set matches and settle mapping elements between object and relationship sets in target and source schemas.

3.1 Terminological Relationships

The meaning of element names provides a clue about which elements match. To match element names, we use WordNet [Fel98, Mil95] which organizes English words into synonym and hypernym sets. Other researchers have also suggested using WordNet to match attributes (e.g. [BCV99, CA99]), but have given few, if any, details.

Initially we investigated the possibility of using 27 available features of WordNet in an attempt to match a token A appearing in the name of a source schema element s with a token B appearing in the name of an target schema element t. The C4.5-generated decision tree, however, was not intuitive.[3] We therefore introduced some bias by selecting only those features we believed would contribute to a human's decision to declare a potential attribute match, namely (f0) same word (1 if $A = B$ and 0 otherwise), (f1) synonym (1 if "yes" and 0 if "no"), (f2) sum of the distances of A and B to a common hypernym ("is kind of") root (if A and B have no common hypernym root, the distance is defined as a maximum number in the algorithm), (f3) the number of different common hypernym roots of A and B, and (f4) the sum of the number of senses of A and B. For our training data we used 222 positive and 227 negative A-B pairs selected from attribute names found in database schemas, which were readily available to us, along with synonym names found in dictionaries. Figure 3 shows the resulting decision tree. Surprisingly, neither f0 (same word) nor f1 (synonym) became part of the decision rule. Feature f3 dominates—when WordNet cannot find a common hypernym root, the words are not related. After f3, f2 makes the most difference—if two words are closely related to the same hypernym root, they are a good potential match. (Note that f2 covers f0 and f1 because both identical words and direct synonyms have zero distance to a common root; this helps mitigate the surprise about f0 and f1.) Lastly, if the number of senses is too high (f4 $>$ 11), a pair of words tends to match almost randomly; thus the C4.5-generated

[3] An advantage of decision-tree learners over other machine learning (such as neural nets) is that they generate results whose reasonableness can be validated by a human.

```
f3 <= 0: NO (222.0/26.0)
f3 > 0
|   f2 <= 2: YES (181.0/3.0)
|   f2 > 2
|   |   f4 <= 11
|   |   |   f2 <= 5: YES (15.0/5.0)
|   |   |   f2 > 5: NO (14.0/6.0)
|   |   f4 > 11: NO (17.0/2.0)
```

Figure 3: Generated WordNet Rule

rule rejects these pairs and accepts fewer senses only if pairs are reasonably close (f2 <= 5) to a common root.

The parenthetical numbers (x/y) following "YES" and "NO" for a decision-tree leaf L give the total number of training instances x classified for L and the number of incorrect training instances y classified for L. Based on the trained decision rule in Figure 3, we compute a confidence value, denoted $conf_1(s,t)$, where s is a source schema element and t is a target schema element. However, we want the feature f0 (same word) to dominate the others and assign a perfect confidence value (1.0) for two tokens if f0 holds. When schema element names are abbreviations, we expand them so that WordNet can recognize them. If the names of both s and t are single-word tokens, the computation of $conf_1(s,t)$ is straightforward based on the decision rule when f0 does not hold. For a "YES" leaf L, we compute confidence factors by the formula $(x\text{-}y)/x$ where x is the total number of training instances classified for L and y is the number of incorrect training instances classified for L. For a "NO" leaf, the confidence factor is $1\text{-}(x\text{-}y)/x$, which converts "NO's" into "YES's" with inverted confidence values. If a schema element name is a phrase instead of a single-word token, we select nouns from the phrase. Then if either s or t has a name consisting of multiple noun tokens, we use an injective greedy match algorithm in Figure 4 to locate the potential matching tokens between the name phrases of s and t. We compute $conf_1(s,t)$ as the average of the confidence values collected from the potential matching tokens obtained from the injective greedy algorithm.

Assuming Schema 1 in Figure 1(a) is a target schema, and Schema 2 in Figure 1(b) is a source schema, when we apply the test for terminological relationships of schema element names, the confidence value $conf_1(s,t)$ is high for the matches such as $(Agent, agent)$, $(Bedrooms, beds)$, $(Bathrooms, baths)$, $(Phone, phone\_day)$, and $(Phone, phone\_evenining)$, as it should be. However, the confidence of $(Golf\_course, location\_description)$ is low, even though "Golf_course" around a house property is a kind of "location_description"; and the confidences of $(Style, category)$, $(Style, location\_description)$, and $(Style, address)$ are high based on the WordNet hierarchical structure, even though they are not semantically correspondent with each other; but, as we shall see, other techniques can sort out this anomaly.

3.2 Data-Value Characteristics

Whether two sets of data have similar value characteristics provides another a clue about which elements match. Previous work in [LC00] shows that this technique can successfully help match elements by considering such characteristics as string-lengths and alphabetic/non-alphabetic ratios of alphanumeric data and means and variances of numerical data. We use features similar to those in [LC00], but generate a C4.5 decision rule rather than a neural-net decision rule. Based on the decision rule, which turns out to be lengthy but has a form similar to the decision tree in Figure 3, we generate a confidence value, denoted $conf_2(s,t)$, for each element pair (s,t) of schema elements

```
Input: a matrix M of confidence values, and a threshold T.
Output: a set of matching attribute pairs.

While there is an unsettled confidence value in M greater than T
    Find the largest unsettled confidence value V in M;
    Settle V by setting it to 1;
    Mark V as being settled;
    For each unsettled confidence value W in the rows and columns of V
        Settle W by setting it to 0;
        Mark W as being settled;
Output the settled attribute pairs whose value is 1;
```

Figure 4: Injective-Match Settling Algorithm

that have data values available.

Testing the decision rule using data values associated with Schema 1 in Figure 1(a) as a source schema and Schema 2 in Figure 1(b) as a target schema, the confidence value $conf_2(s,t)$ is high for the matches such as $(beds, Bedrooms)$, $(baths, Bathrooms)$, $(phone\_day, Phone)$, and (fax, Fax) as expected. However, MLS in the target and *address* in the source tend to look alike according to the value characteristics measured, a surprise which needs other techniques to find the difference. Interestingly, the house location description in *location_description*, the category in *category* , and the house address in *address* of the source schema do not have similar value characteristics with style values in *Style* of the target schemas; this is because either their string length ratios or their alpha/non-alpha ratios are vastly different, as they should be.

3.3 Expected Data Values

Whether expected values appear in a set of data provides yet another a clue about which elements match. For a specific application, we can specify a lightweight domain ontology [ECJ+99], which includes a set of concepts and relationships among the concepts, and associates with each concept a set of regular expressions that matches values and keywords expected to appear for the concept. Then, using techniques described in [ECJ+99], we can extract values from sets of data associated with source and target elements and categorize their data-value patterns based on the regular expressions declared for application concepts. The derived data-value patterns and the declared relationship sets among concepts in the domain ontology can help discover both direct and indirect matches for schema elements.

We declare the concepts and relationship sets in our lightweight domain ontologies independently of any target and source schemas. We call them lightweight for two reasons. (1) The construction of concepts and relationships is not the same as the construction of a conceptual schema in GAV data-integration approaches [Ull97] for integrating heterogeneous information sources. A

GAV data-integration system maintains a global schema, and the system needs to update the global schema when new information sources become available. Thus, the GAV approach requires that the global schema should be complete in the sense that it embodies all the contents in the underlying information sources. We neither require nor expect that the knowledge declared in an application domain ontology is complete for the application. Moreover, (2) the objective of the regular expressions declaring expected values for application concepts is to discover corresponding concepts, not to extract items of interest [ECJ+99]. Since the domain ontology need not be as complete nor as exact as the declarations for a data-extraction ontology, we see our domain ontologies as being lightweight.

Figure 5 shows three components in our real-estate domain ontology, which we used to automate matching of the two schemas in Figure 1 and also for matching real-world schemas in the real-estate domain in general. The three components include an address component specifying *Address* as potentially consisting of *State*, *City*, *County*, and *Street*;[4] a phone component specifying *Phone* as a possible superset of *Day Phone*, *Evening Phone*, *Home Phone*, *Office Phone*, and *Cell Phone*;[5] and a lot-feature component specifying *Lot Feature* as a possible superset of *View* values and individual values *Water Front* and *Golf Course*.[6] Behind a dotted box (or individual value), a regular-expression recognizer [ECJ+99] describes the expected data values for a potential application concept. The ontology explicitly declares that (1) the expected values in *Address* match with a concatenation of the expected values for *Street*, *County*, *City* and *State*; (2) the set of values associated with *Phone* is a superset of the values in *Day Phone*, *Evening Phone*, *Home Phone*, *Office Phone*, and *Cell Phone*; and (3) the set of values associated with *Lot Feature* is a superset of the values associated with the set of *View* values and the singleton-sets *Water Front* and *Golf Course*.

[4] Filled-in (black) triangles denote aggregation ("part-of" relationships).
[5] Open (white) triangles denote generalization/specialization ("ISA" supersets and subsets).
[6] Large black dots denote individual objects or values.

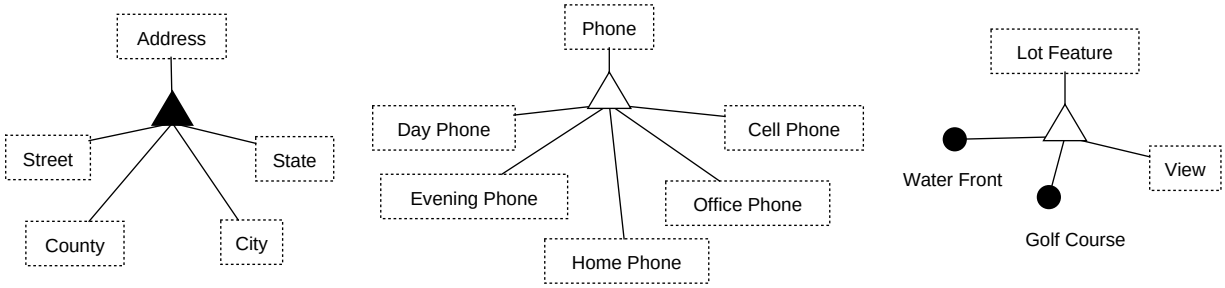


Figure 5: Application Domain Ontology (Partial)

Provided with the domain ontology just described and a set of data values for elements in Schema 1 in Figure 1(a) and Schema 2 in Figure 1(b), we can discover indirect matches as follows. (We first explain the idea with examples and then more formally explain how this works in general.)

1. *Composition* and *Decomposition.* Based on the *Address* declared in the ontology in Figure 5, the recognition-of-expected-values technique [ECJ+99] can help detect that (1) the values of *address* in Schema 1 of Figure 1(a) match with the ontology concept *Address*, and (2) the values of *Street*, *City*, and *State* in Schema 2 of Figure 1(b) match with the ontology concepts *Street*, *City*, and *State* respectively. Thus, if Schema 1 is the source and Schema 2 is the target, we can use *Decomposition* over *address* in the source to derive three virtual object sets such that the three virtual object sets match with *Street*, *City*, and *State* respectively in the target. If we switch and let Schema 2 be the source and Schema 1 be the target, based on the same information, we can identify an indirect match that declares a virtual object set derived by applying *Composition* operation over the source to merge values in *Street*, *City*, and *State* directly match with *address* in the target[7].

2. *Union* and *Selection.* Based on the specification of the regular expression matched for *Phone*, the schema elements *phone_day* and *phone_evening* in Schema 1 of Figure 1(a) match with the

[7]When applying the manipulation operations over sources in data integration applications, the data integration system requires routines to merge/split values so that correctly retrieving data from sources.

concepts *Day Phone* and *Evening Phone* respectively, and *Phone* in Schema 2 of Figure 1(b) also matches with the concept *Phone*. *Phone* in the ontology explicitly declares that the set of expected values of *Phone* is a superset of the expected values of *Day Phone* and *Evening Phone*. Thus, we are able to identify the indirect matching schema elements between *Phone* in Schema 2 and *phone_day* and *phone_evening* in Schema 1. If Schema 1 is the source and Schema 2 is the target, we can apply a *Union* operation over Schema 1 to derive a virtual schema element *Phone'*, which can directly match with *phone* in Schema 2. If Schema 2 is the source and Schema 1 is the target, we may be able to recognize keywords such as *day-time*, *day*, *work phone*, *evening*, and *home* associated with each listed phone in the source. If so, we can use a *Selection* operation to sort out which phones belong in which specialization (if not, a human expert may not be able to sort these out either).

3. *Schema Element Name as Value.* Because regular-expression recognizers can recognize schema element names as well as values, the recognizer for *Lot Feature* recognizes names such as *Water_front* and *Golf_course* in Schema 2 of Figure 1(b) as values. Moreover, the recognizer for *Lot Feature* can also recognize data values associated with *location_description* in Schema 1 of Figure 1(a) such as "*Mountain View*", "*City Overlook*", and "*Water-Front Property*". Thus, when Schema 1 is the source and Schema 2 is the target, whenever we match a target-schema-element name with a source *location_description* value, we can declare "Yes" as the value for the matching target concept. If, on the other hand, Schema 2 is the source and Schema 1 is the target, we can declare that the schema element name should be a value for *location_description* for each "Yes" associated with the matching source element.

We now more formally describe these three types of indirect matches. Let c_i be an application concept, such as *Street*, and consider a concatenation of concepts such as *Address* components. Suppose the regular expression for concept c_i matches the first part of a value v for a schema

element and the regular expression for concept c_j matches the last part of v, then we say that the concatenation $c_i \circ c_j$ matches v. In general, we may have a set of concatenated concepts C_s match a source element s and a set of concatenated concepts C_t match a target element t. For each concept in C_s or in C_t, we have an associated hit ratio. Hit ratios give the percentage of s or t values that match (or are included in at least some match) with the values of the concepts in C_s or C_t respectively. We also have a hit ratio r_s associated with C_s that gives the percentage of s values that match the concatenation of concepts in C_s, and a hit ratio r_t associated with C_t that gives the percentage of t values that match the concatenation of concepts in C_t. To obtain hit ratios for Boolean fields recognized as schema-element names, we distribute the schema-element names over all the Boolean fields.

We decide if s matches with t directly or indirectly by comparing C_s and C_t. If C_s equals C_t, we declare a *direct* match (s, t). Otherwise, if $C_s \supset C_t$ ($C_s \subset C_t$), we derive an *indirect* match (s, t) through a *Decomposition* (*Composition*) operation. If both C_s and C_t contain one individual concept c_s and c_t respectively, and if the values of concept c_s (c_t) are declared as a subset of the values of concept c_t (c_s), we derive an *indirect* match (s, t) through a *Union* (*Selection*) operation. When we have schema-element names as values, distribution of the name over the Boolean value fields converts these schema elements into standard schema elements with conventional value-populated fields. Thus, no additional comparisons are needed to detect direct and indirect matches when schema-element names are values. We must, however, remember the Boolean conversion for both source and target schemas to correctly derive indirect matches.

We compute the confidence value for a mapping (s, t), which we denoted as $conf_3(s, t)$, as follows. If we can declare a direct match or derive an indirect match through manipulating *Union*, *Selection*, *Composition*, *Decomposition*, *Boolean*, and *DeBoolean* for (s, t), and the hit ratios r_s and r_t are above an accepted threshold, we output the highest confidence value 1.0 for $conf_3(s, t)$.

Otherwise, we construct two vectors v_s and v_t whose coefficients are hit ratios associated with concepts in C_s and C_t. To take the partial similarity between v_s and v_t into account, we calculate a VSM [BYRN99] cosine measure $cos(v_s, v_t)$ between v_s and v_t, and let $conf_3(s,t)$ be $(cos(v_s, v_t) \times (r_s + r_t)/2)$.

3.4 Structure

We consider structure as one more technique that provides a clue about which elements match. The former discussed matching techniques compute confidence measures for an element pair between two schemas in element-level, which means determining matching elements in a target schema with an element in a source schema [RB01] in isolation. Our solution takes structural characteristics into account, match combinations of schema elements that appear together in a schema, and compute structure-level mapping elements [RB01].

The former three matching techniques including the terminological relationships, the value characteristics, and the expected data values compare only object sets between a target schema and a source schema. In additional to object set matches, structure matching applies schema structural properties to resolve relationship set matches between two schemas. The object set matches themselves are not enough to provide access paths for retrieving data from the source. Assume that we let Schema 1 of Figure 1(a) as a target schema and Schema 2 of Figure 1(b) as a source schema. Based on the terminological relationships, value characteristics, and expected data values, we obtain the object set matches such as (MLS, MLS) and ($beds$, $Bedrooms$). Without relationship set matches, however, it is impossible to correctly answer a user query "finding houses with 4 bed rooms". In a source-to-target mapping, a mapping element $\omega(t \sim s \Leftarrow \theta_s(\Sigma_S))$ is either an object set match or a relationship set match. Since the mapping element declares that a source schema element s, which is either an element in the source or a view querying over the source, is semantically equivalent to a target element t, we can access data of s into the target for the facts of

t. Intuitively, a source-to-target mapping between a target schema and a source schema describes all of the possible access paths to retrieve data facts from the source into the target.

When comparing structural properties of a target schema and a source schema, we apply a top-down strategy. At the top level, we compute the semantic correspondences between abstracted components of the target and source schemas. Each of the components for both schemas composes of a set of object sets and relationship sets among the object sets. The structure matching algorithm dynamically decides component composition for target and source schemas based on schema structural constraints and available confidence values for potential object set matches from the terminological relationship sets, the value characteristics, and the expected data values. Then, at the bottom level, with the guide of compatible components between the target and source schemas, we compute the finer-level correspondences between the object and relationship sets.

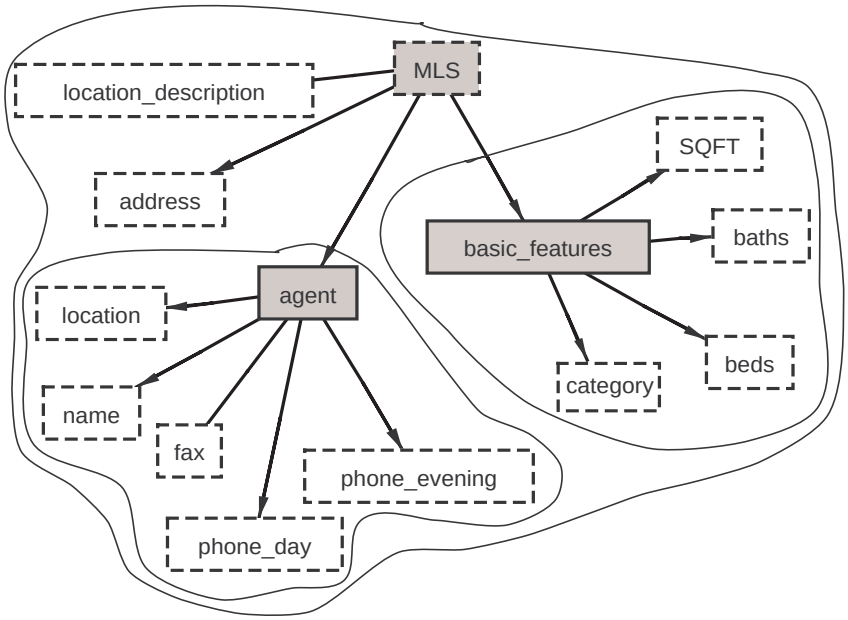
The abstraction for target and source schemas uses equivalence-class transformations [Emb98]. Based on the relationship-set constraints in a conceptual schema H, we analyze H into a set of *equivalence classes* and relationship sets among the equivalence classes. Equivalence-class transformations capitalize on the idea that we may be able to find two or more object sets, or sets of object sets, that are in a one-to-one correspondence and thus in an equivalence class. Let X and Y be subsets of the object sets in H, and let F be a set of functional dependencies over H. X and Y are *equivalent* if $X \rightarrow Y \ \in F^+$ and $Y \rightarrow X \ \in F^+$. If $X \rightarrow Y$ and $Y \rightarrow X$, we write $X \leftrightarrow Y$. The relation $\leftrightarrow$ over subsets of the set of object sets O_H is an equivalence relation because the relation is reflexive, symmetric, and transitive. For any equivalence relation formed from the relation $\leftrightarrow$, we can form a set of pairwise nonintersecting sets of object sets, where each set of object sets determines every other set functionally. We call this set an *equivalence class*. An equivalence class is *trivial* if it only contains a single object set. Otherwise, the equivalence class is *nontrivial*. Based on the definition of equivalence class, a singular set containing an object set o is in an equivalence

class because $\{o\} \leftrightarrow \{o\}$ holds. In our approach, we are interested in *maximum-set* equivalence classes. If an equivalence class e is a maximum-set equivalence class in a schema schema H, there is not other subset e' of O_H in Σ_H available such that $e' \leftrightarrow e$ and $e' \supset e$.[8]he equivalence classes mentioned following in this paper are maximum-set equivalence classes By analyzing a schema H, we can divide the object sets into subsets, each of which form a maximum-set equivalence class, and the subsets do not overlap[Emb98]. We denote the set of maximum-set equivalence classes in H as E_H, where the union of equivalence classes in E_H is the object sets O_H. In Schema 2 of Figure 1(b), there is one non-trivial equivalence class $\{House,\ MLS\}$ containing both $House$ and MLS two object sets. All the equivalence classes in Schema 1 and other equivalence classes in Schema 2 are trivial, singleton sets.

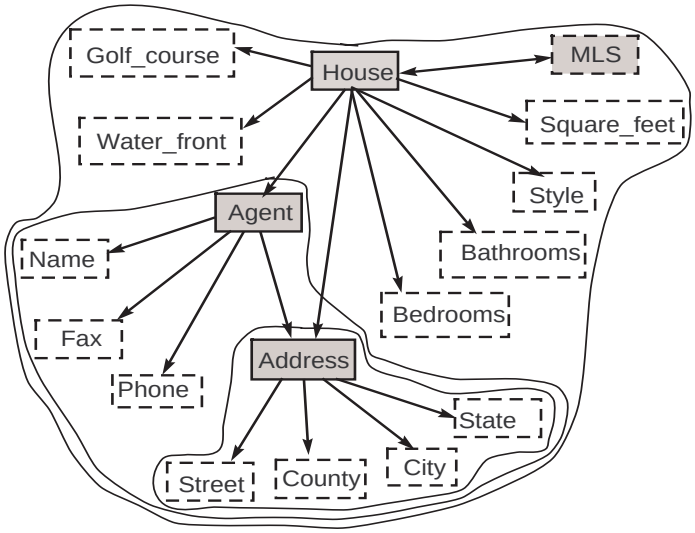
With the equivalence classes in a conceptual schema H, we further analyze a conceptual schema H by abstracting the equivalence classes into a set of *representative* equivalence classes, which we denote as E_H^R. Intuitively, the set of representative equivalence classes of a conceptual schema is the most important and informative ones to describe the purpose of the schema. We distinguish representative equivalence classes E_H^R from E_H, which contains all of equivalence classes in a conceptual schema H, if and only if an equivalence class e satisfies either (1) the equivalence class e is nontrivial, which contains more than one object set, or (2) the object sets in e determine one other set of object sets in an equivalence class e', where $e' \in E_H$. In Figure 6, the shaded boxes describe the object sets in the representative equivalence classes $\{MLS\}$, $\{agent\}$, and $\{basic\_features\}$ in Schema 1 of Figure 1(a), and the representative equivalence classes $\{House,\ MLS\}$, $\{Agent\}$, and $\{Address\}$ in Schema 2 of Figure 1(b).

In additional to representative equivalence classes, the structure matching technique makes use of one other notion, "context" of schema elements. By taking relationship sets around a

[8]T



(a) Schema 1



(b) Schema 2

Figure 6: Context Analysis for Schema 1 and Schema 2 at the First Phase

representative equivalence class e into account, we cluster a set of object sets and relationship sets with a representative equivalence class as a component to describe the conceptual schema. We call this component as the context for the representative equivalence class e, which we denote as $Cont_e$. The context $Cont_e$ for a representative equivalence class e composes a set of object sets, which we denote as $Cont_e^O$, and a set of relationship sets, which we denote as $Cont_e^R$.

To obtain a context $Cont_e$ for a representative equivalence class e in either the target or the source schema, the structure matching technique proceeds in two phases using an context-collection algorithm in Figure 7. In the first phase, we use the the available confidence measures and structural properties in an individual schema to compute contexts for representative equivalence class independently. In the algorithm, since the compatibilities between equivalence classes in the two input schemas are unknown, the algorithm uses a default initialized "false" value, which means that the corresponding equivalence class in this schema is not compatible with any equivalence class in the other schema. Figure 6 shows the contexts of representative equivalence classes for Schema 1 and Schema 2, where each circled curve denotes a context for a representative equivalence class. Note that the shaded boxes inside a circled curve represent the object sets in an equivalence classes. Because the context collection in the algorithm of Figure 7 uses a recursive computation in the for loop based on functional closure, it is possible that the contexts of different representative equivalence classes exist overlap in a schema. For example, the context of the equivalence class $\{MLS\}$ subsumes the contexts of $\{agent\}$ and $\{basic\_features\}$ in Schema 1, and the context of the equivalence class $\{House,\ MLS\}$ subsumes the context of $\{Agent\}$ and the context of $\{Address\}$ in Schema 2.

After the first phase of context computation for representative equivalence classes in target and source schemas, we compare the representative equivalence classes as well as their contexts between the schemas. Thus, we make a guess to map the source schema into the target schemas

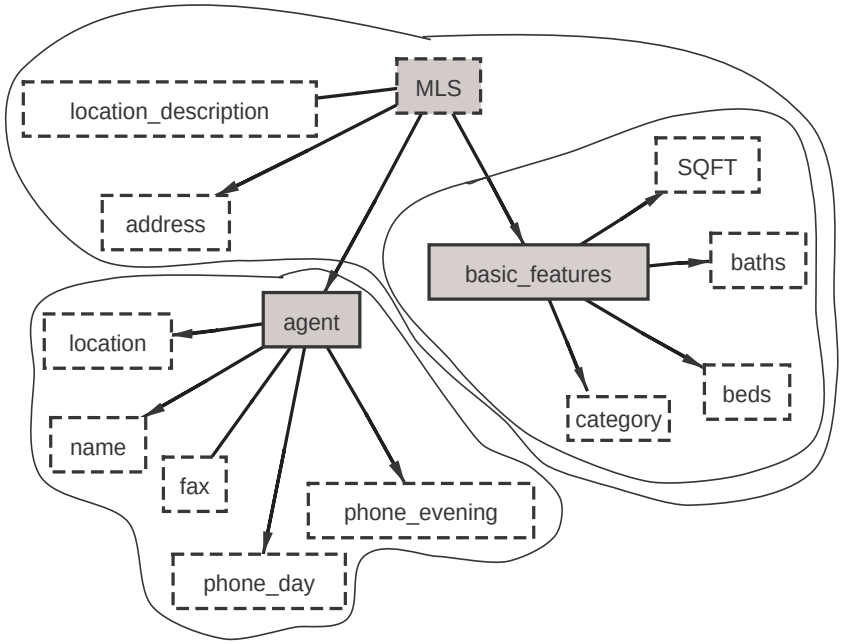
by determining a set of *compatible* representative equivalence classes that are highly similar. In Figure 6, we guess that $\{House,\ MLS\}$ matches $\{MLS\}$, and $\{Agent\}$ matches with $\{agent\}$. Then, in the second phase of context computation for representative equivalence classes, with the available guess about the map between the target and source schemas, we use the algorithm in Figure 7 to recompute contexts for representative equivalence classes in the target and source schemas. Figure 8 shows the new contexts for representative equivalence classes for schemas in Figure 1. Note that the contexts for representative equivalence classes $\{MLS\}$ and $\{House,\ MLS\}$ are smaller than their original contexts before the re-computation. Even though $\{MLS\} \rightarrow \{agent\}$ and $\{House,\ MLS\} \rightarrow \{Agent\}$, however, because $\{agent\}$ and $\{Agent\}$ in the two schemas are correspondent with each other, we do not include their contexts for $\{MLS\}$ and $\{House,\ MLS\}$. We reduce the context scopes for representative equivalence classes at the top-level such that the structure matching limits the search of finer-level correspondences in smaller search spaces.

With the guide of the available comparative representative equivalence classes, instead of globally identifying semantic correspondences between schema elements in target and source schemas, the structure matching technique discovers object and relationship set matches between the contexts of compatible representative equivalence classes. We base our structure contexts for both direct and indirect matches on four intuitive ideas, which we illustrate using Schemas 1 and 2 in

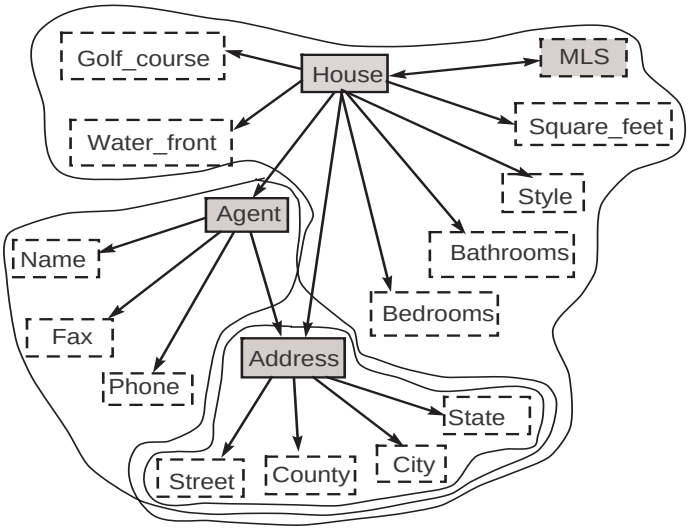
```
Input: a representative equivalence class e in a schema H.
Output: a context C of e.

Include all the object sets in e into a set Co
for each equivalence class e' connected by e
    if e' is non-representative
        Include the object set in e' into Co
    else if e' is not potential compatible
        Include the object sets in the context of e' into Co
Collect relationship sets among Co in H into a set Cr
Output Co and Cr as the context C for e;
```

Figure 7: Context Computation of Representative Equivalence Classes



(a) Schema 1



(b) Schema 2

Figure 8: Context Analysis for Schema 1 and Schema 2 at the Second Phase

Figure 8.

1. *Non-lexical object set.* Two non-lexical object sets match if their element names are similar and the lexical object sets around them describe similar data in the two schemas. A non-lexical object set has only object identifiers in a target or source schema. The object identifiers themselves do not describe the objects in the non-lexical object set. Instead, the values of object sets around the non-lexical object set describe non-lexical objects. The context analysis provides a limited scope for selecting the lexical object sets around the non-lexical object sets. In Figure 8, both the *agent* in Schema 1 and the *Agent* in Schema 2 represent the agent for a house. The confidence value $conf_1(agent,\ Agent)$ computed based on terminological relationships between the two element names declares that *Agent* of Schema 2 is similar to *agent* of Schema 1 in Figure 8. The data associated with the adjacent *Name*, *Fax*, and *Phone* as well as the adjacent *Street*, *City*, *State* around the object *Address* together represent semantics for *Agent* in Schema 2. In contrast, the data associated with the adjacent *location*, *name*, *fax*, *phone_day*, *phone_evening* together represent the semantics of *agent* in Schema 1. By taking the adjacent lexical object sets into account, *Agent* of Schema 2 does match with *agent* of Schema 1.

   One non-lexical object set is possible to match a virtual non-lexical object set between target and source schemas. With the available compatibility between $\{House,\ MLS\}$ of Schema 2 and the equivalence class $\{MLS\}$ of Schema 1 in Figure 8 obtained at the top level of comparison between the schemas, assume that Schema 1 is a source schema and Schema 2 is a target schema, we create a virtual non-lexical object set *House'* based on values for *MLS* in Schema 1. The virtual non-lexical object set $House'$ matches with *House* in Schema 2. The non-lexical object set match satisfy the requirement, for example, the terminological relationships between names and the similarity between values of closely related lexical object

sets, for non-lexical object set matches discussed in the above paragraph.

2. *Lexical object set.* The closely related, for example, adjacent and sibling, lexical and non-lexical object sets supply additional constraints for lexical object sets. In Figure 8(a), *address* and *location* denotes house and agent locations respectively. Based on the context analysis for the two schemas, we got the context distribution for representative equivalence classes of schemas in Figure 8. Figure 8(a) shows that the object set *location* is in the context for $\{agent\}$ and *address* is in the context of $\{MLS\}$. Thus, even though both *address* and *location* describe addresses, we distinguish the semantic correspondences in Figure 8(b) for *location* and *address* by considering the context similarities. In Figure 8(b), the *address* objects are in contexts for $\{House,\ MLS\}$ and $\{Agent\}$. Since we obtained the guess that $\{House,\ MLS\}$ matches $\{MLS\}$, given that the values for *Street*, *City* and *State* describe addresses in Figure 8(b), we decide that there exists an indirect match between the addresses in the context of $\{House,\ MLS\}$ of Figure 8(b) and the addresses in the context of $\{MLS\}$ of Figure 8(a). Similarly, we decide the other indirect match between the addresses in the context of $\{Agent\}$ of Figure 8(b) and the addresses in the context of $\{MLS\}$ of Figure 8(a).

3. *Relationship set.* Each relationship set, which is either in the source or in the target, is a graph, which nodes and edges represent object sets and connections among the object sets respectively. A source relationship set s, which could be a virtual element, matches with a target relationship set t if and only if two graphs signaturing the two relationship sets s and t are isomorphic. That means, a mapping function f_r from s to t exists such that there is an object set $f_r(o_s)$ with constraint c connected by t if and only if there is an object set o_s with constraint c connected by s. Thus, the requirement for relationship set matches falls into two categories: (1) type requirements to satisfy node isomorphism, and (2)constraint requirements

to satisfy edge isomorphism. The type requirement between two nodes is satisfied if and only if there exists an object set match $\omega(o_t \sim o_s \leftarrow \theta_{o_s}(\Sigma_S))$ where the two nodes represents o_s and o_t. To check constraint compatibility between two connections, [BE03] proposed four cases, which guide user's involvements for schema mapping operations while translating source data into the target.

Since our approach for schema mapping allows derived data in source schemas, the exhaustive search for relationship set matches between Σ_T and V_S, where T is a target schema and S is a source schema, would be exponential time complexity. To avoid generating a large amount of views over a source schema, we make constriction to the search space for view generation. At the bottom-level comparison of the structure matching, With the obtained compatible contexts, the matching technique first discovers object set matches. Then ,with the guide of object set correspondences, it discovers relationship set matches. Intuitively, we want to use type requirements for relationship set matches to trigger deriving views over source within a subset of a context, where the subset composes of a set of object sets and a set of relationship sets among the object sets. The decision of the object sets is based on obtained object set matches and context limitations. For example, we let Schema 1 in Figure 8(a) as a target schema and Schema 2 in Figure 8(b) as a source schema. Assume that we discover that the objects in *Agent* in the context of $\{Agent\}$ corresponds the objects in *agent* in the context of $\{agent\}$, and discover that the values for *Street*, *County*, *City*, and *State* of objects *Address* in the context of $\{Agent\}$ in Schema 2 semantically corresponds *location* in the context of $\{agent\}$ in Schema 1. To obtain the semantic correspondence of the relationship set $agent-location$ in Schema 1, with the available semantic correspondences between object sets in the contexts of $\{agent\}$ and $\{Agent\}$, we derive views over $Agent-Address$, $Address-Street$, $Address-County$, $Address-City$, and *Address-State* in the source to form a virtual

relationship set matching $agent - location$ in the target.

4 Matching Algorithm

We have implemented an algorithm using our matching techniques that produces both direct and indirect matches between a source schema S and a target schema T. We use one running example applying two schemas in Figure 1. Let Schema 1 be a source schema S, and let Schema 2 be a target schema T.

Step 1: *Compute conf measures between S and T.* For each pair of schema elements (s, t), which are either both lexical object sets or both non-lexical object sets, the algorithm computes a confidence value, $conf(s,t)$, to combine the output confidence values of the three nonstructural matching techniques. We compute $conf(s,t)$ using the following formula.

$$conf(s,t) = \begin{cases} conf_1(s,t)\ , \text{ if } s \text{ and } t \text{ are non-lexical object sets} \\ 1.0\ , \text{ if } conf_3(s,t) = 1.0 \text{ and } s \text{ and } t \text{ are lexical object sets} \\ w_s(conf_1(s,t)) + w_v(conf_2(s,t) + conf_3(s,t))/2\ , \text{ otherwise} \end{cases}$$

In this formula, w_s and w_v are experimentally determined weights.[9] When the confidence value $conf_3(s,t) = 1.0$, we let $conf_3$ dominate and assign $conf(s,t)$ as 1.0 and keep the detected manipulation operations (*Selection*, *Union*, *Composition*, *Decomposition*, *Boolean*, *DeBoolean*) for indirect element matches. The motivation for letting $conf_3(s,t)$ dominate is that when expected values appear in both source and target schema elements and they both match well with the values we expect, this is a strong indication that the elements should match (either directly or indirectly). Since the domain ontology is not guaranteed to be complete (and may even have some inaccuracies) for a particular application domain, the confidence values obtained from the other techniques can complement and compensate for the inadequacies of the domain knowledge. This motivates the third part of the computation for $conf(s,t)$.

Step 2: *Analyze equivalence classes and their semantic correspondences between S and T.* We identify two sets of equivalence classes E_S and E_T in S and T using an algorithm in [Emb98]. We

[9] The two parameters w_s, which weights schema element names, and w_v, which weights schema element values, are application dependent. Using a heuristic guide, however, we can determine the two parameters based on schemas and available data even without experimental evidence. If the schema element names are informative and the data is not self descriptive, we assign w_s as 0.8 and w_v as 0.2. On the other hand, if the schema element names are not informative and the data is semantically rich, we assign w_s as 0.2 and w_v as 0.8. For all other cases, we assign both w_s and w_v as 0.5.

further distinguish the representative equivalence classes between S and T, and compute their original contexts using the algorithm in Figure 7. Figure 6 shows the original contexts for representative equivalence classes for the schemas in Figure 1.

When comparing two representative equivalence classes e_S and e_T, where $e_S \in E_S$ and $e_T \in E_T$, we take three factors into account: (1) a set of combined confidence measures $\{conf(s,t)|s \in e_S,\ t \in e_T\}$, (2) an importance similarity measure $sim_{importance}(e_S, e_T)$, and (3) a vicinity similarity measure $sim_{vicinity}(e_S, e_T)$. We can declare a compatible pair, which we denote as $(e_T \sim e_S)$ if there exists at least one $conf(s,t)$, $sim_{importance}(e_S, e_T)$, and $sim_{vicinity}(e_S, e_T)$ are high. The latter two measures together represent the similarity between the contexts of e_S and e_T, which we denote as $Cont_{e_S}$ and $Cont_{e_T}$ respectively obtained using algorithm in Figure 7. Given an experimentally determined threshold, th_{conf},[10] we calculate $sim_{importance}(e_S, e_T)$ and $sim_{vicinity}(e_S, e_T)$ based on the following formulas.

$$sim_{vicinity}(s,t) = max(\quad \frac{|\{x|x \in Cont^O_{e_S} \wedge \exists y \in Cont^O_{e_T}(conf(x,y) > th_{conf})\}|}{|Cont^O_{e_S}|}, \\ \frac{|\{x|x \in Cont^O_{e_T} \wedge \exists y \in Cont^O_{e_S}(conf(y,x) > th_{conf})\}|}{|Cont^O_{e_T}|})$$

$$sim_{importance}(e_S, e_T) = 1.0 - |\frac{|Cont_{e_S}|}{|\Sigma_S|} - \frac{|Cont_{e_T}|}{|\Sigma_T|}|$$

Intuitively, $sim_{vicinity}$ measures the similarity of the vicinity surrounding e_S and the vicinity surrounding e_T, and $sim_{importance}$ measures the similarity of the "importance" of e_S and the "importance" of e_T where we measure the "importance" of an equivalence class e by counting the number of schema elements in the context of e. When the number of schema elements is largely different, it is difficult to decide the vicinity similarity based on one singular measure, $sim_{vicinity}$ [MBR01]. The conceptual analysis techniques discussed in [CAFP98] motivated $sim_{importance}$, which helps measure the context similarity from an additional perspective.

The comparison between equivalence classes in the target T and the source S provides a guess about the semantic correspondences. By using the algorithm in Figure 7, we recompute the contexts for compatible equivalence classes. Figure 8 shows the modified contexts for representative

[10] For any application, the computed confidence values tend to converge to a specific high measure for element matches between two schemas. Thus, we use a universal threshold value. Experimentally, we have determined that 0.8 works well across all applications.

equivalence classes in our running example. At this step, we finish the top-level comparison between S and T , and we are ready to detect the object and relationship set matches at the bottom-level.

Step 3: *Discover object and relationship set matches.* For each matching pair ($e_T \sim e_S$) of representative equivalence classes settled in Step 2, we first settle object set matches between $Cont^O_{e_S}$ and $Cont^O_{e_T}$ for e_S and e_T that match with high confidence ($conf = 1.0$). For all remaining unsettled object sets of $Cont^O_{e_S}$ and $Cont^O_{e_T}$, we find a best possible match using the injective algorithm in Figure 4 so long as the confidence of the match is above the threshold, th_{conf}. For each of the matches, given the expected-value matches, we keep the information required to transform source elements into virtual elements that directly match with target object sets with the recognized object set matches. For example, we keep the *Decomposition* operations identified by the expected data values between *location* of *agent* in Schema 1 and *Street*, *County*, *City*, and *State* in Schema 2. We are going to use the operations to specify mapping expressions for indirect matches at Step 4.

With the available semantic correspondences between object sets in S and T, we further discover matches between relationship sets. The recognition for most of relationship set matches are also limited in the contexts of compatible representative equivalence classes between S and T at this step. However, for the relationship sets among contexts of representative equivalence classes, we identify semantic correspondences globally without the limitation of contexts. Within the context for a representative equivalence class e, we distinguish the relationship sets in the context $Cont_e$ for an equivalence class e into *inner relationship sets*, which are relationship sets among object sets in e, and outer relationship sets, which are other relationship sets in $Cont_e$ that are not inner relationship sets. We denote $Cont^{R_i}_e$ as the set of inner relationship sets and $Cont^{R_o}_e$ as the set of outer relationship sets.

Within the contexts of two compatible representative equivalence classes e_T and e_S, first, we recognize the semantic correspondences for inner relationship sets $Cont^{R_i}_{e_T}$ in the context of the target representative equivalence class e_T with relationship sets or views over $Cont_{e_S}$, the context of e_S in the source. For example, we use *Skolemization* to derive a virtual relationship $House' - MLS$ and a virtual object set $House'$ based on the values for MLS in the context of $\{MLS\}$ in the source, Schema 1 of Figure 8(a), to match with $House - MLS$ and $House$ in the context of $\{House,\ MLS\}$ in the target, Schema 2 of Figure 8(b). Second, we recognize the semantic correspondences for outer relationship sets $Cont^{R_o}_e$ in the context of the target representative equivalence class e_T.

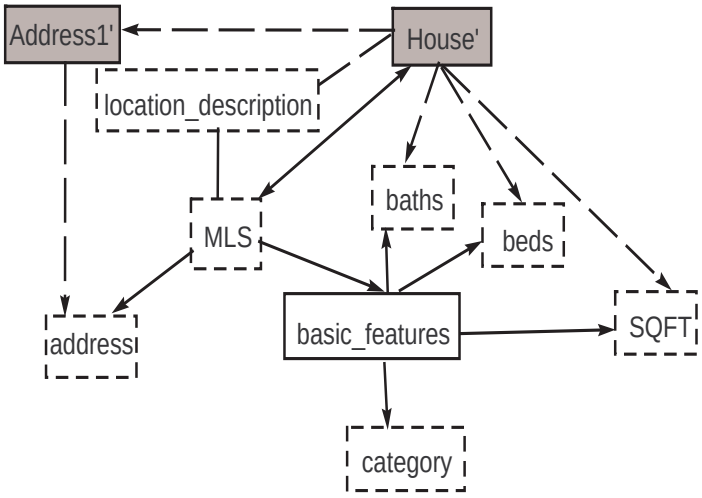
For example, to match with $House - Bedrooms$, which is a outer relationship set in the context of $\{House,\ MLS\}$ in the target, we use *Join* and *Projection* to derive a virtual relationship set $House' - beds$ over the context of $\{MLS\}$ in the source. Figure 9(a) and Figure 9(b) show the virtual object and relationship sets in the contexts for $\{House,\ MLS\}$ and $\{Agent\}$ obtained when discovering relationship-set correspondences. The dashed lines represent virtual relationship sets, and the shaded boxes represent virtual object sets. Given the value correspondences detected based on expected data values, we keep the information required to transform source elements into virtual elements that directly match with relationship sets with the recognized relationship set matches. For example, in the context of $\{MLS\}$, the relationship sets $Address1' - address$ corresponds the target relationship sets $Address - Street$, $Address - County$, $Address - City$, and *Address-State* based on *Composition* operation recognized by the expected values. The obtained *Composition* operation is transferred from the semantic correspondences between object sets *address* in the source context of $\{MLS\}$ and *Street*, *County*, *City*, and *State* in the target context of $\{House,\ MLS\}$.

After discovering relationship set matches within contexts of compatible representative equivalence classes, we further discover the relationship set matches for inter relationship sets between contexts. In our running example, a target relationship set $House - Agent$ is an inter relationship sets between the contexts for $\{House,\ MLS\}$ and $\{Agent\}$ in Figure 8(b). With the available object set match between $House'$ in the source and $House$ in the target, where $House'$ is a virtual object set derived when processing the inner relationship set $House - MLS$ in the context of target representative equivalence class $\{House,\ MLS\}$, and the other object set match between *agent* in the source and *Agent* in the target, we derive a virtual relationship set $House' - agent$ over the source in Figure 9(c) beyond the limitation of contexts.

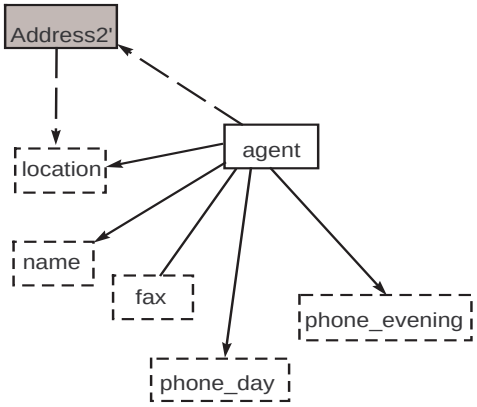
In the following, the $\Leftarrow$ explained in each step denotes a derivation of a virtual schema element on the left through applying the algebra expression on the right. The derivation describes the view definitions triggered by object set correspondences while recognizing relationship set matches within and beyond context limitations.

1. *Deriving virtual object and relationship sets in the context of* $\{MLS\}$.

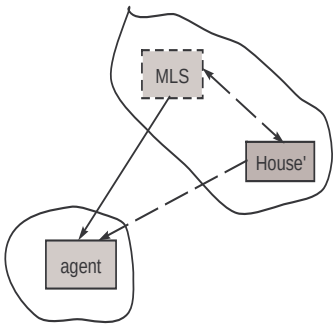
   $House' - MLS \Leftarrow \varphi_{f_{House'}}(MLS)$
   $House' \Leftarrow \pi_{House'}(House' - MLS)$
   $House' - Address1' \Leftarrow \varphi_{f_{Address1'}}(House')$
   $Address1' \Leftarrow \pi_{Address1'}(House' - Address1')$



(a) In the Context of $\{MLS\}$



(b) In the Context of $\{Agent\}$



(c) Inter Relationship Sets between the contexts of $\{Agent\}$ and $\{MLS\}$

Figure 9: Discover Object and Relationship Set Matches

$Address1' - address \Leftarrow \pi_{Address1',address}(MLS - House' \bowtie House' - Address1' \bowtie MLS - address)$
$House' - baths \Leftarrow \pi_{House',baths}(MLS - basic\_features \bowtie basic\_features - baths \bowtie House' - MLS)$
$House' - beds \Leftarrow \pi_{House',beds}(MLS - basic\_features \bowtie basic\_features - beds \bowtie House' - MLS)$
$House' - SQFT \Leftarrow \pi_{House',SQFT}(MLS - basic\_features \bowtie basic\_features - SQFT \bowtie House' - MLS)$
$House' - location\_description \Leftarrow \pi_{House',location\_description}(MLS - location\_description \bowtie House' - MLS)$

The structure matching decides that the values for MLS, *bath*, *beds*, $SQFT$ in the context of $\{MLS\}$ of Figure 8(a) directly corresponds to the values for MLS, $Bathrooms$, $Bedrooms$, $Square\_feet$ in the context of $\{House,\ MLS\}$ of Figure 8(b) respectively. In additional to the direct object set matches, the algorithm determines that the values for *location_description* in the context of $\{MLS\}$ are generalization of the lot description implied in the values for $Water\_front$ and $Golf\_course$ in the target context of $\{House,\ MLS\}$, and the values for *Street*, *County*, *City*, and *State* in the target are split values for values in *address* in the source.

The derivation of virtual relationship sets $House' - MLS$ happened when processing inner relationship set $House - MLS$ in the target context of $\{House,\ MLS\}$. When applying the *Skolemization* operator to derive virtual object set *House'*, the system can compute the skolem function $f_{House'}$ as a function on MLS values in MLS because the target object set MLS, which is semantically equivalent to MLS in the source, is functionally dependent on $House$, which is semantically equivalent to $House'$. When processing the the outer relationship set $House - Address$ in the target, we use one other *Skolemization* operation to derive a virtual relationship set $House' - Adderess'$, which is based on the functional terms in the derived virtual object set $House'$. The matching technique derives other virtual relationship sets when processing outer relationship sets in the context of $\{House,\ MLS\}$.

2. *Deriving virtual object and relationship sets in the context of* $\{agent\}$.

$agent - Address2' \Leftarrow \varphi_{f_{Address2'}}(agent)$
$Address2' \Leftarrow \pi_{Address2'}(agent - Address2')$
$Address2' - location \Leftarrow \pi_{Address2',location}(agent - Address2' \bowtie agent - location)$

The structure matching decides that the objects in *agent* and the values for *name*, fax in the context of $\{agent\}$ of Figure 8(a) directly corresponds to the objects in *Agent* and the values for $Name$, Fax in the context of $\{Agent\}$ of Figure 8(b) respectively. With

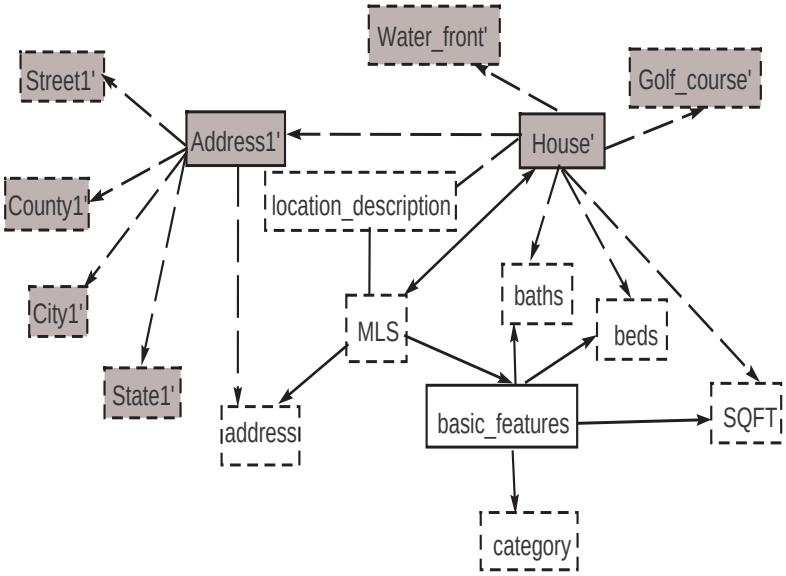
the object set matches, the relationship sets $Agent - Name$, $Agent - Fax$ in the target directly match $agent - name$, $agent - fax$ in the source. Thus, it is no need to derive virtual relationship sets in the source to correspond with the two target relationship sets. In additional to the direct object set matches, the algorithm determines that the values for *Phone* in the context of $\{Agent\}$ are generalization of the phone numbers for *phone_day* and *phone_evening* in the context of $\{agent\}$, and the values for *Street*, *County*, *City*, and *State* in the context of $\{Agent\}$ are split values for values in *location* in the context of $\{agent\}$. The indirect object set matches guide the derivation of virtual relationship sets $agent - Phone'$ and $Address2' - location$ over the context of $\{agent\}$.

3. *Deriving a virtual relationship set between the contexts of* $\{MLS\}$ *and* $\{agent\}$.
   $House' - agent \Leftarrow \pi_{House', agent}(House - MLS' \bowtie MLS - agent)$

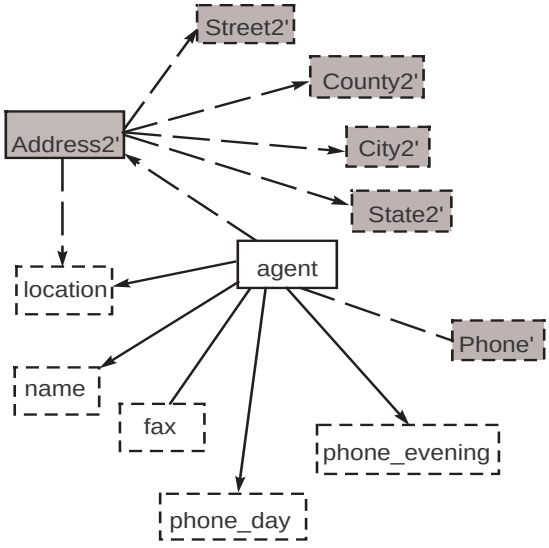
   Even though the view derivation for inter-relationship-set matches is beyond the limitation of contexts of compatible representative equivalence classes, we, however, still can constrict the search space with the guide of object set matches.

Step 4: *Specify mapping expression for object and relationship set matches.* For direct matches, the specification of mapping expressions for mapping elements are straightforward. However, the specification of mapping expressions for indirect matches is not trivial. Within this step, we use a bottom-up strategy to derive mapping expressions for indirect matches. At the bottom level, we derive virtual elements based on instance-level information with the limitation of contexts for compatible representative equivalence classes between S and T. Then, at the top level, we derive virtual elements globally beyond the limitation of contexts. We discuss the the two levels as follows.

First, *use instance-level information to derive virtual object and relationship sets.* In our matching framework, we apply instance-level information for two matching techniques including data-value characteristics and expected data values. However, the data-value characteristics does not contribute to indirect matches. Thus, the derivation of virtual object and relationship sets applying instance-level information dependent on confidence measures output from expected data values. Figure 10 shows the virtual object and relationship sets derived after applying the instance-level information in the running example. We still use the shaded boxes to denote virtual object sets, and use the dashed lines to denote virtual relationship sets.



(a) In the context of $\{MLS\}$



(b) In the context of $\{Agent\}$

Figure 10: Deriving Virtual Object and Relationship Sets based on Expected Data Values

1. *Deriving virtual object and relationship sets in the context of* $\{MLS\}$.

$House' - Golf\_course' \Leftarrow \beth^{\text{“}Yes\text{”},\text{“}No\text{”}}_{location\_description,Golf\_course'}(House' - location\_description)$
$Golf\_course' \Leftarrow \pi_{Golf\_course'}(House - Golf\_course')$
$House' - Water\_front' \Leftarrow \beth^{\text{“}Yes\text{”},\text{“}No\text{”}}_{location\_description,Water\_front'}(House' - location\_description)$
$Water\_front' \Leftarrow \pi_{Water\_front'}(House - Water\_front')$
$Address1' - Street1' \Leftarrow \pi_{Address1',Street1'}(\gamma^{R^{address}_{Street1'}}_{address,Street1'}(Address1' - address))$
$Street1' \Leftarrow \pi_{Street1'}(Address1' - Street1')$
$Address1' - County1' \Leftarrow \pi_{Address1',County1'}(\gamma^{R^{address}_{County1'}}_{address,County1'}(Address1' - address))$
$County1' \Leftarrow \pi_{County1'}(Address1' - County1')$
$Address1' - City1' \Leftarrow \pi_{Address1',City1'}(\gamma^{R^{address}_{City1'}}_{address,City1'}(Address1' - address))$
$City1' \Leftarrow \pi_{City1'}(Address1' - City1')$
$Address1' - State1' \Leftarrow \pi_{Address1',State1'}(\gamma^{R^{address}_{State1'}}_{address,State1'}(Address1' - address))$
$State1' \Leftarrow \pi_{State1'}(Address1' - State1')$

Applying two *DeBoolean* operators makes new virtual relationship sets such that the values in the formed virtual relationship sets uses boolean indicators “Yes” as values. The application of four *DeComposition* operator uses routines $R^{address}_{Street1'}$, $R^{address}_{County1'}$, $R^{address}_{City1'}$, and $R^{address}_{State1'}$ to decompose the string values for *address* as values for the new virtual object sets $Street1'$, $County1'$, $City1'$, and $State1'$.

2. *Deriving virtual object and relationship sets in the context of* $\{agent\}$.

$Address2' - Street2' \Leftarrow \pi_{Address2',Street2'}(\gamma^{R^{location}_{Street2'}}_{location,Street2'}(Address2' - location))$
$Street2' \Leftarrow \pi_{Street2'}(Address2' - Street2')$
$Address2' - County2' \Leftarrow \pi_{Address2',County2'}(\gamma^{R^{location}_{County2'}}_{location,County2'}(Address2' - location))$
$County2' \Leftarrow \pi_{County2'}(Address2' - County2')$
$Address2' - City2' \Leftarrow \pi_{Address2',City2'}(\gamma^{R^{location}_{City2'}}_{location,City2'}(Address2' - location))$
$City2' \Leftarrow \pi_{City2'}(Address2' - City2')$
$Address2' - State2' \Leftarrow \pi_{Address2',State2'}(\gamma^{R^{location}_{State2'}}_{location,State2'}(Address2' - location))$
$State2' \Leftarrow \pi_{State2'}(Address2' - State2')$
$agent - Phone' \Leftarrow \rho_{phone\_day \leftarrow Phone'}(agent - phone\_day) \cup \rho_{phone\_evening \leftarrow Phone'}(agent - phone\_evening)$
$Phone' \leftarrow \pi_{Phone'} agent - Phone'$

Applying four *DeComposition* operator makes virtual relationship sets based on routines $R^{location}_{Street2'}$, $R^{location}_{County2'}$, $R^{location}_{City2'}$, and $R^{location}_{State2'}$ decomposing the string values for *location* as values for the new virtual object sets $Street2'$, $County2'$, $City2'$, and $State2'$. Indeed, the four routines used here are same as those used to extract values for $Street1'$, $County1'$, $City1'$, and $State1'$ in the context of $\{MLS\}$.

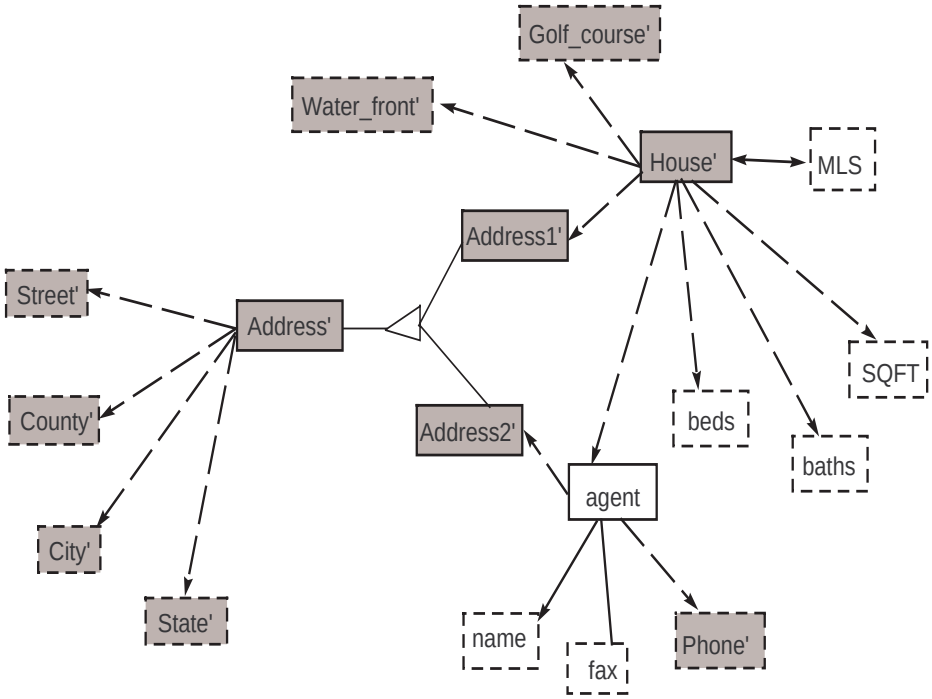


Figure 11: Source elements in the Source-to-Target Mapping between S and T

Second, *use schema-level information to derive virtual object and relationship sets.* The matching techniques apply source and target schema structural characteristics to derive virtual object and relationship sets beyond the constraints of contexts. Basically, we collect matches happened in different contexts. For example, both objects in $Address1'$ in the context of $\{MLS\}$ and objects in $Address2'$ in the context of $\{agent\}$ in the source correspond objects in $Address$ in the target, which is in both the context of $\{House,\ MLS\}$ and $\{Agent\}$. In a source-to-target mapping, between S and T, however, a target element $t \in \Sigma_T$ corresponds to at most one source element $s \in V_S$. Thus, we use *Union* or *Selection* operations to force the 1-1 relationship sets between target elements in Σ_T and source elements in V_S.

1. *Deriving virtual object sets for indirect object set matches.*

$$Address' \Leftarrow \rho_{Address1' \leftarrow Address'} Address1' \cup \rho_{Address2' \leftarrow Address'} Address2'$$
$$Street' \Leftarrow \rho_{Street1' \leftarrow Street'} Street1' \cup \rho_{Street2' \leftarrow Street'} Street2'$$
$$County' \Leftarrow \rho_{County1' \leftarrow County'} County1' \cup \rho_{County2' \leftarrow County'} County2'$$
$$City' \Leftarrow \rho_{City1' \leftarrow City'} City1' \cup \rho_{City2' \leftarrow City'} City2'$$
$$State' \Leftarrow \rho_{State1' \leftarrow State'} State1' \cup \rho_{State2' \leftarrow State'} State2'$$

```
Input: a virtual source element s and a set of derivation rules Views.
Output: a mapping expression Me of s.

Include the derivation rule for s in Views into Me
for each virtual element s' appeared in the rule
    Derive a mapping expression Me' of s'
    Union Me with Me'
```

Figure 12: Derivation of a Mapping Expression for an Virtual Element

When the objects in $Address1'$ and $Address2'$ are generalized in $Address'$, it is an object identity problem. In this paper, we leave this problem out and assume that we already have a resolution. At this level, if *Selection* operation is applied, we may be able to recognize keywords for values in an object set to sort out the specializations. If not, a human expert may not be able to sort these out either.

2. *Deriving virtual relationship sets for indirect relationship set matches.*

$$
\begin{aligned}
Address' - Street' &\Leftarrow \rho_{Address1' \leftarrow Address', Street1' \leftarrow Street'} Address1' - Street1 \\
&\quad \cup \rho_{Address2' \leftarrow Address', Street2' \leftarrow Street'} Address2' - Street2' \\
Address' - County' &\Leftarrow \rho_{Address1' \leftarrow Address', County1' \leftarrow County'} Address1' - County1 \\
&\quad \cup \rho_{Address2' \leftarrow Address', County2' \leftarrow County'} Address2' - Street2' \\
Address' - City' &\Leftarrow \rho_{Address1' \leftarrow Address', City1' \leftarrow City'} Address1' - City1 \\
&\quad \cup \rho_{Address2' \leftarrow Address', City2' \leftarrow City'} Address2' - City2' \\
Address' - State' &\Leftarrow \rho_{Address1' \leftarrow Address', State1' \leftarrow State'} Address1' - State1 \\
&\quad \cup \rho_{Address2' \leftarrow Address', State2' \leftarrow State'} Address2' - State2' \\
\_agent - Phone' &\leftarrow \sigma_{Phone'}(agent - Phone')
\end{aligned}
$$

The open white triangle in Figure 11 denotes generalization/specialization. We use the notion to illustrate that the objects in $Address'$ are union of the objects in $Address1'$ and $Address2'$. The last derivation forces the participation constraint of *agent* in the relationship set $\_agent - Phone'$ matches with the participation constraint "1:1" of *Agent* in the relationship set $Agent - Phone$. We can use the list of issues in [BE03] to resolve the constraint matching problem.

Figure 11 shows the source elements in the source-to-target mapping between S and T of our running example. We use a table *Views* memorize the derivation rules for virtual elements in V_S. The derivation algorithm in Figure 12 has a recursive flavor to collect a mapping expressions for a virtual source element of an indirect match. Note that the derivation of virtual object and

relationship sets is incrementally added as the matching algorithm proceeds, thus, the algorithm in Figure 12 to compute mapping expressions for virtual elements in the source must finish after finite steps.

Step 5: *Output both direct and indirect element matches with manipulation operations.*

5 Data Integration

The following five characteristics describe TIQS. Each characteristic addresses one of the five major issues we want to address for data integration.

1. *Heterogeneity.* Each relation in a target schema, which is our global schema, is predefined and independent of any source schema. Moreover, we wrap sources in isolation, without reference to the global schema.[11] Thus, in TIQS, source and target schemas use different structures and vocabularies. A mapping tool [XE03] within TIQS produces a set of *mapping elements* semi-automatically in a source-to-target mapping that maps a source schema to a target schema. The mapping elements include both direct and many indirect semantic correspondences. Thus, TIQS reduces heterogeneity by applying semantic correspondences expressed using source-to-target mappings between target and source schemas.

2. *Scalability.* Although TIQS still requires a DBA to validate and sometimes adjust the generated source-to-target mappings, TIQS largely automates this mapping procedure [XE03]. This facilitates scalability, and allows TIQS to specify views over a large number of source schemas that match with elements in the target schema in a semi-automatic way.

3. *Continual infusion and change of local information sources.* When a new information source becomes available (changes), a source-to-target mapping must be created (adjusted). With the assistance of the semi-automatic mapping tool in TIQS, the maintenance requires little manual work to create (adjust) mappings.

4. *Query processing complexity.* Whenever a users poses queries in terms of target relations, TIQS uses its generated source-to-target mappings to reduce query reformulation to simple rule unfolding (standard execution of views in ordinary databases). This reduces query processing complexity.

[11] Often these sources are structured, and we simply take the local schema without change [ETL02].

5. *Evolution.* If the target schema evolves, the mapping tool of TIQS semi-automatically generates (or adjusts) mapping elements between the new target schema and the source schemas.

TIQS operates in two phases: design and query processing. In the design phase, the system synergistically automates the generation of source-to-target mappings. For a mapping element $\omega(t \sim s \Leftarrow \theta_s(\Sigma_S))$, we think the source element s is a *virtual target-view element* for t. This leads automatically to a rewriting of every target element as a union of corresponding virtual target-view elements. In the query processing phase, a user poses queries in terms of target relations. Query reformulation thus reduces to rule unfolding by applying the view definition expressions for the target relations in the same way database systems apply view definitions.

5.1 The Data Integration System

Definition 1. A *data-integration system* I is a triple $(T, \{S_i\}, \{M_i\})$, where T is a target schema, $\{S_i\}$ is a set of n source schemas, and $\{M_i\}$ is a set of n source-to-target mappings, such that for each source schema S_i there is a mapping M_i from S_i to T, $1 \leq i \leq n$.

The source and target schemas in I are model instances in OSM-L. If the wrappers in I use different expression languages beyond OSM-L, we use import programs to bridge the representations used by wrappers and the internal representation for schemas.

To specify the semantics of I, we start with a *valid interpretation* D_{S_i} of a source schema $S_i \in \{S_i\} \in I$, $1 \leq i \leq n$. For an interpretation of a schema H to be valid, each tuple in D_H must satisfy the constraints specified for H. In our running example, let S_1 denote the information source whose schema is Schema 1 in Figure 1(a). For purposes of illustration, assume that we have one additional information source, which we denote as S_2, and assume that the wrapped Schema for S_2 is exactly the same as Schema 2 in Figure 1(b). Further, assume that both S_1 and S_2 have valid interpretations. The tables in Figure 13 show some partial populated data for relations in S_1 and S_2.

A target interpretation D_{S_iT} with respect to a source interpretation D_{S_i} in I (1) is a valid interpretation of T, and (2) satisfies the mapping M_i between S_i and T with respect to D_{S_i}. Assume that the mapping function for M_i is f_i. If f_i matches s_k with t_j, c is a tuple for t_j in D_{S_iT} if and only if c is a tuple for s_k derived through applying the mapping expression $\theta_{s_k}(\Sigma_{S_i})$ over

house	location_description
h10	Water_front
h10	Golf_course
h11	Golf_course

house	basic_features
h10	b10
h11	b11

basic_features	SQFT
b10	1000
b11	1500

(a) Partial populated data for relations in S_1

House	Water_front
h1	Yes
h2	No
h3	Yes

House	Golf_course
h1	Yes
h2	Yes
h3	No

House	Square_feet
h1	990
h2	1420
h3	2500

(b) Partial populated data for relations in S_2

Figure 13: Some partial populated data for relations in sources

D_{S_i}. For our running example, recall that the target schema is Schema 1 in Figure 1(a). Section 4 describes the source-to-target mapping between the target schema and the source schema for S_1 in Figure 1(a). Based on the source-to-target mapping, a valid target interpretation D_{S_1T} with respect to a valid interpretation D_{S_1} contains tuples in a valid interpretation for the schema in Figure 11. Moreover, since we assume that the source schema for S_2 is exactly the same as the target schema T, the source-to-target mapping between the target schema and the source schema for S_2 is trivial. Thus, the tuples in a valid target interpretation D_{S_2T} with respect to D_{S_2} are those of source relations in S_2.

The semantics of I, denoted as $sem(I)$, are defined as follows: $sem(I) = \{D_{S_iT} \mid D_{S_iT}$ is a target interpretation with respect to D_{S_i}, $S_i \in I\}$.[12] Intuitively, the semantics of I represent relevant data allowed in a predefined target schema T retrieved from available heterogeneous information sources. We are able to prove that if a source has a valid interpretation, then we can "load" data from the source into the target such that the part of the target populated from the source will necessarily have a valid interpretation [BE03].

[12]When sources share objects, both the object-identification problem and the data-merge problem need a resolution. (Note that neither this paper nor other papers that focus on data integration with virtual global schemas resolve these problems. The focus of this paper is on mediation, mappings, and query reformulation.)

5.2 Query Reformulation

In the design phase, the data-integration system I collects the information including a target schema T, a set of source schemas $\{S_i\}$, and a set of source-to-target mappings $\{M_i\}$. In the query-processing phase, the system reformulates user queries in polynomial time.

In this paper, a query can be a conjunctive query, a conjunctive query with arithmetic comparisons, or a recursive query. We use logic rule notation in [Ull88] to express user queries. Here, the queries are in terms of elements in Σ_T, which means that a predicate in a query body is either a target relation in Σ_T or a head predicate of a logic rule. We call a predicate representing a target relation appearing in a query body a *target predicate*. Like pure DataLog, we do not allow negations of predicates that appear in user queries because we adopt the "Open World Assumption" in our approach. In our running example, assume that a user wants an answer to the query, "For houses on water-front and golf-course property, list the number of square feet." We can express this query, which we denote as $q_{example}$, using the following logic rule.

$$\begin{aligned} House-Square\_feet(x,y) \ :- \ & House-Square\_feet(x,z) \ \bowtie \ House-Golf\_course(x,\text{“}Yes\text{”}) \\ & \bowtie \ House-Water\_front(x,\text{“}Yes\text{”}) \end{aligned}$$

Let q be a user query such as the one above. When evaluating q over $sem(I)$, the system I transparently reformulates q as q^{Ext}, which is a query evaluated by retrieving data from the underlying information sources in I. Let $D = \{D_{S_i} | S_i \in \{S_i\} \in I\}$ be the set of valid interpretations of source schemas in I. By reformulating q as q^{Ext}, the system transforms the task of evaluating q over $sem(I)$ into a task of answering q^{Ext} over D.

The system I reformulates a user query q by applying the inclusion dependencies for target relations collected in the design phase. Since a user poses queries in terms of elements in Σ_T, each target relation r_i that appears in the body of q corresponds to a set of inclusion dependencies ID_i, $1 \leq i \leq N$ and $N = |\Sigma_T|$. We expand the user query q for each inclusion dependency $(S_j.s_k \ \subseteq \ r_i)$

in ID_i, where $s_k \in V_{S_j}$ ($1 \leq j \leq n$ and n is the number of sources), and where r_i is a target predicate that appears in the body of q, by adding a logic rule, $r_i(\overline{X}) \ :- \ S_j.s_k(\overline{X})$, where $\overline{X}$ is a vector of variables. Thus, the added logic rules as well as the logic rule of q together form the query q^{Ext}.

To reformulate $q_{example}$ for our running example, the system I applies inclusion dependencies for target relations $House-Square\_feet$, $House-Water\_front$ and $House-Golf\_course$ appearing in the body of query $q_{example}$. In addition to the logic rule above for the user query $q_{example}$, the following logic rules are added to form the reformulated query $q^{Ext}_{example}$.

$$
\begin{array}{l}
House-Square\_feet(x,y) \ :- \ S_1.House'-SQFT(x,y) \\
House-Square\_feet(x,y) \ :- \ S_2.House-Square\_feet(x,y) \\
House-Water\_front(x,u) \ :- \ S_1.House'-Water\_front'(x,u) \\
House-Water\_front(x,u) \ :- \ S_2.House-Water\_front(x,u) \\
House-Golf\_course(x,v) \ :- \ S_1.House'-Golf\_course'(x,v) \\
House-Golf\_course(x,v) \ :- \ S_2.House-Golf\_course(x,v)
\end{array}
$$

To evaluate a reformulated query q^{Ext} over sources, the system I decomposes q^{Ext} into sub-queries, and retrieves and combines query answers to sub-queries from individual information sources. We use a logic program P^{Ext}_D to describe the evaluation of q^{Ext} over D, where D is the set of valid interpretations of source schemas in I. The logic program P^{Ext}_D is defined as follows.

- *Rules.* The logic rules for q^{Ext}.
- *Facts.* For each source relation $S_j.s_k$ in the body of the logic program for q^{Ext}, we treat data for the source relations as ground facts. For example, if a tuple t is in the source relation $S_j.s_k$, we have the fact[13]: $S_j.s_k(t)$.

By evaluating P^{Ext}_D, the facts for the head predicate of q are query answers to the reformulated query q^{Ext} over D, which we denote as q^{Ext}_D. Note that when sending a sub-query to obtain data

[13]We use this logic program to capture the semantics of q^{Ext}_D. In real-world applications, query processing in I can optimize the evaluation of q^{Ext}.

from an information source S_j, the system I also sends the mapping expression $\theta_{s_k}(\Sigma_{S_j})$ to the source S_j so that the source S_j correctly executes the mapping expression to derive source facts for s_k. Given the data in Figure 13, for our running query example, $q_{example}$, we add the following list of source facts from S_1 and S_2 to the rules in $q^{Ext}_{example}$ above to form a logic program $P^{Ext}_{exampleD}$.

$S_1.House' - Water\_front'(h10, \text{“}Yes\text{”})$
$S_1.House' - Golf\_course'(h10, \text{“}Yes\text{”})$
$S_1.House' - Golf\_course'(h11, \text{“}Yes\text{”})$
$S_1.House' - Square\_feet(h10, 1000)$
$S_1.House' - Square\_feet(h11, 1500)$
$S_2.House - Water\_front(h1, \text{“}Yes\text{”})$
$S_2.House - Water\_front(h2, \text{“}No\text{”})$
$S_2.House - Water\_front(h3, \text{“}Yes\text{”})$
$S_2.House - Golf\_course(h1, \text{“}Yes\text{”})$
$S_2.House - Golf\_course(h2, \text{“}Yes\text{”})$
$S_2.House - Golf\_course(h3, \text{“}No\text{”})$
$S_2.House - Square\_feet(h1, 990)$
$S_2.House - Square\_feet(h2, 1420)$
$S_2.House - Square\_feet(h3, 2500)$

Note that the facts for S_1 have been transformed according to the source-to-target mapping in Section 4. By evaluating this logic program $P^{Ext}_{exampleD}$, we obtain the $House - Square\_feet$ facts $(h1, 990)$ and $(h10, 1000)$ in $q^{Ext}_{exampleD}$.

With query reformulation in place, we can now prove that query answers to any query are *sound*—every answer to a user query is a fact according to the semantics of I—and *maximal*—the query answers contain all the facts the sources have to offer with respect to the facts allowed in the global target schema. Let q_I denote the query answers to a user query q over the semantic of I, $sem(I)$, which represents all data relevant to the target schema T from all information sources in I. The proofs are based on an observation that the semantics of q_I can be captured using a logic program P_I. The logic program P_I is defined as follows.

- *Rule.* A user query q in terms of target relations in T.
- *Facts.* For each tuple t for a target relation r in D_{S_jT}, where $D_{S_jT} \in sem(I)$ and $S_j \in \{S_j\} \in I$, we have the fact: $r(t)$. (Note that these facts include all facts the sources have for T.)

The facts for the head predicate of q by evaluating P_I are the query answers q_I. Based on the characteristics of the two logic programs P_D^{Ext} and P_I, we can now prove the following theorem.

Theorem. *Let $I = (T, \{S_i\}, \{M_i\})$ be a data-integration system. Let $D = \{D_{S_i} | S_i \in \{S_i\} \in I\}$ be the set of valid interpretations of source schemas in I. Let q_I be the query answers obtained by evaluating q over $sem(I)$ and let q_D^{Ext} be the query answers obtained by evaluating q^{Ext} over D. Given a user query q in terms of target relations, a tuple $a = < a_1, a_2, \ldots, a_M >$ where M is the number of variables and constants in the head predicate of q, is in q_I if and only if a is a tuple in q_D^{Ext}.*

Proof. Assume that we define two logic programs P_D^{Ext} and P_I based on the semantics of q_D^{Ext} and q_I respectively as we discussed in Section **??**.

If. Assume that a tuple $a = < a_1, a_2, \ldots, a_M >$ is in q_D^{Ext} but not in q_I. Since a is in q_D^{Ext}, there exists a substitution ϑ, which binds variables in P_D^{Ext} with constants such that the evaluation of $P_D^{Ext}\vartheta$ yields the tuple a for the head predicate of q. By using a subset of substitution ϑ' of ϑ for variables in q while evaluating P_I, since a is not in q_I, there must exist at least one subgoal of q that is not satisfied in $P_I\vartheta'$. Based on the evaluation theory in [SC90], the subgoal could be a target predicate or an arithmetic comparison. We make an analysis for each possibility as follows.

- *Arithmetic comparison.* Assume that we have an arithmetic comparison, which is a subgoal of q, satisfied in $P_D^{Ext}\vartheta$ but not in $P_I\vartheta'$. Since we use the same bindings for variables of q in ϑ and ϑ', if the arithmetic comparison is satisfied in $P_D^{Ext}\vartheta$, it must also be satisfied in $P_I\vartheta'$. Thus, the unsatisfied subgoal is not an arithmetic comparison.

- *Target predicate.* Assume that we have a target predicate r, which is a subgoal of q, satisfied in $P_D^{Ext}\vartheta$ but not in $P_I\vartheta'$. Based on the substitution ϑ, the subgoal r is satisfied in P_D^{Ext} because $r(c)$ holds, where c is a vector of constants using the bindings in ϑ. Since $r(c)$ holds,

there must exist a rule $r(c) \ :- \ S_j.s_k(c)$ and a fact $S_j.s_k(c)$, where $S_j \in \{S_i\} \in I$ and $s_k \in V_{S_j}$, in $P_D^{Ext}\vartheta$ such that $r(c)$ is derived from the rule and the fact. Since we define the rule, $r(c) \ :- \ S_j.s_k(c)$, based on an inclusion dependency ($S_j.s_k \ \subseteq \ r$), there must exist a mapping element ($r \sim S_j.s_k \Leftarrow \theta_{s_k}(\Sigma_{S_j})$) between S_j and T. Since s_k matches with r and c is a tuple of s_k in S_j, based on the semantics of D_{S_jT} in $sem(I)$, c is a tuple of r in D_{S_jT}. Then, based on the definition of P_I, there must exist a ground fact $r(c)$ in P_I. Since $r(c)$ is a ground fact in P_I, the subgoal $r(c)$ is satisfied in $P_I\vartheta'$. This is contrary to the assumption. Thus, the unsatisfied subgoal in P_I must not be a target predicate.

By analyzing the two possibilities, we conclude that all of the subgoals in q are satisfied while evaluating $P_I\vartheta'$ to obtain the tuple a as a fact for the head predicate of q. Thus, based on the semantics of P_I, a is a tuple in q_I. This is contrary to our assumption.

Only if. Assume that we have a tuple $a = < a_1, a_2, \ldots, a_M >$ in q_I but not in q_D^{Ext}. Since a is a tuple in q_I, there must exist a substitution ϑ for variables in q such that the evaluation of $P_I\vartheta$ outputs the tuple a as a fact for the head predicate of q. By using the same substitution ϑ for variables in q while evaluating P_D^{Ext}, since a is not in q_D^{Ext}, there must exist at least one subgoal in q that cannot be satisfied in $P_D^{Ext}\vartheta$. Based on the evaluation theory in [SC90], the subgoal could be a target predicate or an arithmetic comparison. We make an analysis for each possibility as follows.

- *Arithmetic comparison.* Assume that we have an arithmetic comparison, which is a subgoal of q, satisfied in $P_I\vartheta$ but not in $P_D^{Ext}\vartheta$. Since we use the same bindings for variables of q in ϑ, if the arithmetic comparison is satisfied in $P_I\vartheta$, it must also be satisfied in $P_D^{Ext}\vartheta$. Thus, the unsatisfied subgoal is not an arithmetic comparison.

- *Target predicate.* Assume that we have a target predicate r, which is a subgoal of q, satisfied

in $P_I\vartheta$ but not in $P_D^{Ext}\vartheta$. Since the subgoal r is satisfied in $P_I\vartheta$, $r(c)$ holds, where c is a vector of constants by using bindings in ϑ. Since $r(c)$ holds, based on the definition of P_I, c is a tuple of r in a target interpretation D_{S_jT} of $sem(I)$, where $S_j \in \{S_j\} \in I$. Based on the semantics of D_{S_jT}, since c is a tuple of r in D_{S_jT}, there must exist a mapping element $(r \sim S_j.s_k \Leftarrow \theta_{s_k}(\Sigma_{S_j}))$ between S_j and T where c is a tuple of the source relation $S_j.s_k$. By applying the mapping element that matches $S_j.s_k$ with r, we can derive an inclusion dependency $(S_j.s_k \subseteq r)$. Hence, since we have the inclusion dependency $(S_j.s_k \subseteq r)$ and the fact that c is a tuple of the source relation $S_j.s_k$, based on the definition of P_D^{Ext}, there must exist a rule $r(\overline{X}) :- S_j.s_k(\overline{X})$ and a fact $S_j.s_k(c)$ in $P_D^{Ext}\vartheta$, where $S_j \in \{S_j\} \in I$, $s_k \in V_{S_j}$, and $\overline{X}$ is a vector of variables corresponding attributes of r. Based on the rule and the fact, $r(c)$ holds in $P_D^{Ext}\vartheta$. This is contrary to the assumption. Thus, the unsatisfied subgoal in $P_D^{Ext}\vartheta$ must not be a target predicate.

By analyzing the two possibilities, we conclude that all of the subgoals in q are satisfied while evaluating $P_D^{Ext}\vartheta$ to obtain the tuple a as a fact for the head predicate of q. Thus, based on the semantics of P_D^{Ext}, a is a tuple in q_D^{Ext}. This is contrary to our assumption. □

6 Experimental Results

We evaluate the performance of our approach based on three measures: precision, recall and the F-measure, a standard measure for recall and precision together [BYRN99]. Given (1) the number of direct and indirect matches N determined by a human expert, (2) the number of correct direct and indirect matches C selected by our process described in this paper and (3) the number of incorrect matches I selected by our process, we compute the recall ratio as $R = C/N$, the precision ratio as $P = C/(C + I)$, and the F-measure as $F = 2/(1/R + 1/P)$. We report all these values as percentages.

We tested the approach proposed here using the running example in our paper and also on several real-world schemas in three different application domains. In our experiments, we evaluated the contribution of different techniques and different combinations of techniques. We always used both structure and terminological relationships because given any two schemas, these techniques always apply even when no data is available. Thus, we tested our approach with four runs on each source-target pair. In the first run, we considered only terminological relationships and structure. In the second run, we added data-value characteristics. In the third run, we replaced data-value characteristics with expected data values, and in the fourth run we used all techniques together.

6.1 Running Example

We applied the matching algorithm explained in Section 4 to the schemas in Figure 1 populated (by hand) with actual data we found in some real-estate sites on the Web. First we let Schema 1 in Figure 1(a) be the source and Schema 2 in Figure 1(b) be the target. Then, we reversed the schemas and let Schema 2 be the source and Schema 1 be the target.

Run Nr.	Number of Matches	Number Correct	Number Incorrect	Recall %	Precision %	F-Measure %
1 (WS)	31	17	2	55%	89%	68%
2 (WCS)	31	17	0	55%	100%	71%
3 (WES)	31	31	0	100%	100%	100%
4 (WCES)	31	31	0	100%	100%	100%

W = Terminological Relationships using WordNet
C = Data-Value Characteristics
E = Expected Data Values
S = Structure

Table 1: Results for Running Example: Source-Schema 1, Target-Schema 2

Table 1 shows a summary of the results for each run in the first test where we let Schema 1 be the source and Schema 2 be the target. In the first run for the first test, the algorithm discovered 8 direct object set matches correctly, but it also misclassified the source object set *address* (meaning house address) and the virtual relationship set $house' - address$ by matching them with the target

schema element *Style* (meaning "apartment" or "townhouse") and $House - Style$. In the first run, the algorithm also successfully discovered 9 of the 23 indirect matches. For example, the algorithm matches the union of values for *phone_day* and *phone_evening* in the source matches with values for *Phone* in the target. By using *Skolemization* operation, the algorithm matches the object identifiers for a virtual non-lexical $House'$ based on the values in MLS matches with object identifiers in *House*. The structure matching algorithm also correctly matches relationship sets, such as $House - Square\_feet$, $House - Bathrooms$, and $House - Bedrooms$, in the target with virtual relationship sets derived in the source based on *Join* and *Projection* operations. Especially, the algorithm uses *Skolemization* operator two times to compute virtual objects in $Address1'$ and $Address2'$ matching with objects in *Address* in the target, and correctly output a *Union* operation to union the two sets of object identifiers in a new virtual object set $Address'$ that directly matches with *Address*. In the second run, by adding the analysis of data-value characteristics, the false positive between *Style* and *address* disappeared, but the algorithm generated no more indirect matches than in the first run. In both the third and fourth runs, the algorithm successfully discovered all direct and indirect matches. Note that we correctly generated a *Selection* operator to select the right subsets of *location_description* (meaning "view," etc.) in Schema 1 for *Water_Front* and *Golf_Course*, and discarded the remaining values, which were inapplicable for Schema 2. The *Selection* operator sorted out values based on the expected data values specified in the lightweight domain ontologies.

Run Nr.	Number of Matches	Number Correct	Number Incorrect	Recall %	Precision %	F-Measure %
1 (WS)	25	17	2	68%	89%	77%
2 (WCS)	25	17	0	68%	100%	81%
3 (WES)	25	25	0	100%	100%	100%
4 (WCES)	25	25	0	100%	100%	100%

W = Terminological Relationships using WordNet
C = Data-Value Characteristics
E = Expected Data Values
S = Structure

Table 2: Results for Running Example: Source-Schema 2, Target-Schema 1

The result of the second test on our running example, in which we switched the schemas and let Schema 2 be the source schema and Schema 1 be the target schema, gave the results as in Table 2.

In the first run for the second test, the algorithm discovered 9 direct and 8 indirect matches correctly, but it also misclassified the source object set *Style* and $House - Style$ by matching them with the target object set *address* and a virtual relationship set $house' - address$. In the first run, We observe, however, that although we correctly generated a *Selection* operator to specialize the *Phone* value in Schema 2, the value transformation for *Selection* depends on keywords such as $day - time$, *day*, *work phone*, *evening*, and *home* associated with listed phone numbers. If the keywords are not available, the *Selection* operation fails to sort out the *phone* values. In the second run, by adding the analysis of data-value characteristics, the false positive between *Style* and *address* disappeared. In both the third and fourth runs, the algorithm successfully discovered all direct and indirect matches. Especially noteworthy, we observed that our approach correctly discovered context-dependent indirect matches (e.g. (*City*, *address*), (*State*, *address*), ...) and appropriately produced operations consisted of a combination of *Composition*, *Join*, *Projection*, and *Selection*. The *Selection* operator sorted out the addresses composed from *state*, *city*, *county*, and *street* based on the two relationship sets $House - Address$ and $Agent - Location$ in Schema 2.

6.2 Real-World Examples

We considered three real-world applications: *Course Schedule*, *Faculty*, and *Real Estate* to evaluate our approach. We used a data set downloaded from the LSD homepage [DDH01] for these three applications, and we faithfully translated the schemas from DTDs used by LSD to rooted conceptual-model graphs. Table 3 shows the characteristics of the source schemas. The table shows the number of object sets and relationship sets (Number of ObjSets and Number of RelSets), the maximum depth of the DTD trees. The rightmost column shows the percentage of object and relationship sets in a source schema that have either direct or indirect matches with other source schemas. The percentages show that the source schemas for *Course Schedule* and *Faculty* are relatively highly matchable, and the source schemas for *Real Estate*, however, are not.

For testing these real-world applications, we decided to let any one of the schema graphs for an application be the target and let any other schema graph for the same application be the source. We decided not to test any single schema as both a target and a source. Since for each application there were five schemas, we tested each application 20 times. All together we tested 60 target-

Domains	Number of Sources	Number of ObjSets	Number of RelSets	Number of Depth	Matchable %
Course Schedule	5	15 - 19	14 - 18	1 - 4	62 - 93 %
Faculty	5	14	13	3	100%
Real Estate	5	34 - 88	33 - 86	1 - 4	17 - 73%

Table 3: Domains and Schemas for Real-World Examples

Application	Number of Matches	Number Correct	Number Incorrect	Recall %	Precision %	F-Measure %
Course Schedule	490	454	6	93%	99%	96%
Faculty	540	540	0	100%	100%	100%
Real Estate	875	816	92	93%	90%	92%
All Applications	1905	1810	98	95%	95%	95%

Table 4: Results for Real-World Examples

source pairs. For each target-source pair, we made four runs, the same four (WS, WCS, WES, and $WCES$) we made for our running example. All together we processed 240 runs. Table 4 shows as summary of the results for the real-world data using all four techniques together.

In *Faculty* application, there were 4 indirect relationship set matches because of constraints conflicts between relationship sets. Because the structure matching algorithm correctly identified the constraints conflicts, for all four runs on *Faculty* every measure (recall, precision, F-measure) was 100%. Since the five source schemas are highly similar, and however, the data instances collected for each object sets are vastly different, we assigned a higher weight for w_S than w_V to dominate schema-level information.

In *Course Schedule* application, there were indirect relationship set matches requiring manipulations using *Join*, *Skolemization* and *Projection* operators. For *Course Schedule*, the first and second run achieved above 90% and below 95% on all measures; and the third and fourth run gave the results for *Course Schedule* as Table 4 shows. When using all the four techniques, the correctly recognized mapping elements included 382 direct and 72 indirect matches. Even when values for lexical object sets are not available, since most of the indirect matches appeared in this application are largely dependent on schema-level information, the matching algorithm correctly identified direct 376 and 76 indirect matches for this application.

The *Real Estate* application exhibited several indirect object and relationship set matches. Overall, the algorithm correctly identified 417 direct and 399 indirect matches. The problem of

Merged/Split Values appeared four times, the problem of *Subsets/Supersets* appeared 48 times, and the problem of *Schema Element Name as Value* appeared 10 times. The experiments showed that the application of expected data values in the third and fourth run greatly affected the performance. In the first run, the measures were only about 75%. In the second run, the use of data-value characteristics improved the performance, but only a little because the measures were still below 80%. By applying expected data values in the last two runs, however, the performance improved dramatically. In the third run, the F-measures reached 91% and reached 92% by using all four techniques as Table 4 shows.

Our process successfully found all the indirect matches related to the problems of *Merged/Split Values* and *Schema Element Name as Value*. For the problem of *Subsets/Supersets*, our process correctly found all of the indirect matches related to 44 of 48 problems of *Subsets/Supersets* and incorrectly declared 4 extra *Subsets/Supersets* problems. Of these eight, six of them were ambiguous, making it nearly impossible for a human to decide, let alone a machine. In four cases there were various kinds of phones for firms, agents, contacts, and phones with and without message features, and in another two cases there were various kinds of descriptions and comments about a house written in free-form text. The two clear incorrect happened when our process unioned/selected office and cell phones together and mapped them to phones for a firm instead of just mapping office phones to firm phones and discarding cell phones, which had no match at all in the other schema.

6.3 Discussion

The experimental results show that the combination of terminological relationships and structure alone can produce fairly reasonable results if schemas are highly matchable and indirect matches happen because of the problem *Path as Relationship Sets*. Moreover, the result shows that, by adding our technique of using expected data values, the performances are dramatically better even for applications, for example, *Real Estate*, whose schemas are relatively complex. Unexpectedly, the technique of using data-value characteristics did not help very much for these application domains. Our analysis of data-value characteristics is similar to the analysis in SEMINT [LC00], which produced good results for their test data. The data instances in the real-world applications we used, however, do not appear to be as regular as might be expected. The statistics are highly variant, for example, in applications such as *Course Schedule* and *Real Estate*. For these applications, a

large amount of training data would be needed to train a universal decision tree required for this approach. Collecting enough training data, however, is not a trivial problem.

Some element matches failed in our approach partly because they are potentially ambiguous, and our assertions about what should and should not match are partly subjective.[14] Even though we tested our approach using the same test data set as in LSD [DDH01], the answer keys were generated separately and LSD focuses on computing direct object set matches. Furthermore, neither the experimental methodologies nor the performance measures used are the same. With this understanding, we remark that they reported approximate accuracies of 70% for *Course Schedule*, 90% for *Faculty*, 70% and 80% for the two experiments they ran on the *Real Estate* application. Thus, although our raw performance numbers are an improvement over [DDH01], we do not try to draw any final conclusion.

One obvious limitation of our approach is the need to construct an application-specific domain ontology. Currently, we manually construct these domain ontologies. As we explained in Section 3, however, these domain ontologies are lightweight and are relatively easy to construct and need not be complete. It is possible, however, to make use of statistical learning techniques to collect a set of informative and representative keywords for application concepts. Thus, without human interaction, except for some labeling, we can make use of many keywords taken from the data of the application itself and thus specify regular-expression recognizers for the application concepts at least in a semi-automatic way. Furthermore, many values, such as dates, times, and currency amounts are common across many application domains and can easily be shared. Since domain ontologies appear to play an important role in indirect matching, finding ways to semi-automatically generate them is a goal worthy of some additional work.

One other limitation of our approach is that the schemas for real-world applications we used in the experiment are in same domains even though they may not have largely overlap[15]. For example, the root nodes, which describe the designated object of primary interest, of the five schemas used in each application are semantically correspondent. The structure matching algorithm we used applies a top-down strategy to compare two schemas. When discovering object and relationship set matches, the algorithm is robust to recognize the correspondences between object sets by ap-

[14] It is not always easy to do ground-truthing [HKL+01].

[15] The matchable percentages in real estate application is only in the range from 17% to 73%

plying available schema-level and data instance-level information. When discovering relationship set matches, however, the algorithm is highly depend on structure similarities at the context-level comparison. As [DDH01] mentioned, this is typically the case for "aggregator" domains, where the data-integration system provides access to sources that offer essentially the same service. In future, we plan to examine more applications beyond such limitation using our algorithm.

7 Related Work

[RB01] provides a survey of several schema mapping systems. And [Ull97] compares the two basic approaches GAV and LAV for data integration. We do not repeat this work here, but instead describe work related to our approach from three perspectives: (1) work on discovering direct matches for schema elements, (2) work on discovering indirect matches for schema elements, and (3) work in integrating data from heterogeneous information sources.

Direct Matches. Most of the approaches [BCV99, DDH01, EJX01, LC00, MBR01, MZ98, PTU00] to automating schema mapping focus only on generating direct matches for schema elements.

- In some of our previous work [EJX01], we experimented with using data instances to help identify direct element matches. In this paper, we refine this work by adding a structural component and also extend it to the harder problem of discovering indirect matches.

- Like our approach, the LSD system [DDH01] applies a meta-learning strategy to compose several base matchers, which consider either data instances, or schema information. LSD largely exploits machine learning techniques. There are two phases in the LSD system: one is training and the other is testing. In the training phase, LSD requires training data for each matching element between two schemas for base matchers and the meta matcher. For each different application, however, both base and meta learners have to be supervised, and the supervisor must supply and mark training data to train the learners. Our approach differs in the three ways. (1) We applied machine learning algorithms only to terminological relationships and data-value characteristics. (2) Our system learned a universal decision tree for all application domains based on a domain-independent training set. Thus our system avoids the work of collecting and labeling training data for each application as in LSD. (3) To

combine techniques, we let structure features guide the matching based on the results from multiple kinds of independent matches.

- SEMINT [LC00] applies neural-network learning to automating schema mapping based on instance contents. It is an element-level schema matcher because it only considers attribute matching without taking the structure of schemas into account.

- The structure matching algorithm in Cupid [MBR01] motivated our structure matching algorithm. Cupid, however, does not properly handle two schemas that are largely different. Moreover, the structure matching algorithm Cupid matches two schemas using a bottom-up strategy. Our matching algorithm discovers direct and indirect matches using a top-down strategy.

- ARTEMIS [BCV99], DIKE [PTU00], and Cupid [MBR01] exploit auxiliary information such as synonym dictionaries, thesauri, and glossaries. All their auxiliary information is schema-level—does not consider data instances. In our approach, the auxiliary information including data instances and domain ontologies provide a more precise characterization of the actual contents of schema elements. The imported dictionary we use, WordNet, is readily available and no work is required to produce thesauri as in other approaches.

Indirect Matches. Some work on indirect matches is starting to appear [BE03, MBR01, MHH00, MWJ99], but researchers are only beginning to scratch the surface of the multitude of problems.

- Both Cupid [MBR01] and SKAT [MWJ99] can generate global $1 : n$ indirect matches [RB01]. To illustrate what this means, if in Figure 1 we let Schema 1 be the source and Schema 2 be the target, and if we make *address* a lexical object set rather than a non-lexical object set and discard *street*, *county*, *city*, and *state* in Schema 2, Cupid can match both *Address* and *Location* in the source directly with the modified *address* in the target. Thus Cupid can generate a global $1 : n$ indirect match through a *Union* operation. Our approach, however, can find indirect matches for *Location* and *Address* in the source with *street*, *county*, *city*, and *state* in the target based on finding expected data values and using the *Decomposition* operator as well as the *Union* operator, something which is not considered in Cupid.

- The Clio system [MHH00] introduces an interactive mapping creation paradigm based on value correspondence that shows how a value of a target schema element can be created from a set of values of source elements. A user or DBA, however, is responsible to manually input the value correspondences.

- [BE03] proposes a mapping generator to derive an injective target-to-source mapping including indirect matches in the context of information integration. The mapping generator raises specific issues for a user's consideration. The mapping generator, however, has not been implemented. Our work therefore builds on and is complimentary to the work in [BE03].

Data Integration. Data integration is thought as one of the most important problems in modern information systems. In this paper we focus on data integration systems without materializing the global schemas. We describe several approaches related to the proposed approach for data integration in this paper.

- [CLL01] surveys the most important query processing algorithms proposed in the literature for LAV [LRO96, GKD97], and describes the principle GAV [CGMH$^+$94] data-integration systems and the form of query processing they adopt. In a GAV approach, query reformulation reduces to rule unfolding. However, changes in information sources or adding a new information source requires a DBA to revise the global schema and the mappings between the global schema and source schemas. Thus, GAV is not scalable for large applications. LAV scales better, and is easier to maintain than GAV because DBAs create a global schema independently of source schemas. Then, for a new (or changed) source schema, the DBA only has to give (adjust) a *source description* that describes source relations as views of the global schema. Automating query reformulation in LAV, however, has exponential time complexity with respect to query and source schema definitions. Thus, LAV has low query performance when users frequently pose complex queries.

- [FLM99] proposes a *Global-Local-as-View* (GLAV) approach, which combines expressive powers of both LAV and GAV. In a GLAV approach, the independence of a global schema, the maintenance to accommodate new sources, and the complexity to reformulate queries are the same as in LAV. However, instead of using a restricted form of first-order logical sentences as

in LAV and GAV to define view definitions, GLAV uses flexible first-order sentences such that it allows a view over source relations to be a view over global relations in source descriptions. Thus, GLAV can derive data using views over source relations, which is beyond the expressive ability of LAV, and it allows conjunctions of global relations, which is beyond the expressive ability of GAV. Our solution, TIQS, also has the ability to derive views over source schemas. The sets of view-creation operators in TIQS, however, are more powerful because GLAV has nothing comparable to merge/split or Boolean operators. Moreover, GLAV claims no ability to semi-automate the specification of source descriptions.

- [CCGL02] proposes a translation algorithm to turn LAV into GAV such that it can keep LAV's scalability and obtain GAV's simple query reformulation. The translation results in a logic program that can be used to answer queries using rule unfolding. However, even though the translation to obtain the logic program is in polynomial time, the evaluation of the logic program could produce an exponential number of facts because of recomputing source relations over all source data. In contrast, TIQS encapsulates views for source relations in mapping elements. Since the view definitions are immediately available, query processing in TIQS has better query performance than the translation approach. Furthermore, [CCGL02] does not claim the ability to semi-automate the specification of source descriptions.

8 Conclusion and Future Work

We presented a framework for automatically discovering both direct matches and many indirect matches between sets of source and target schema elements. In our framework, multiple techniques each contribute in a combined way to produce a final set of matches. Techniques considered include terminological relationships, data-value characteristics, expected values, and structural characteristics. We detected indirect element matches for *Join*, *Projection*, *Selection*, *Union*, *Skolemization*, *Composition*, and *Decomposition* operations as well as *Boolean* conversions for *Schema-Element Names as Values*. We base these operations and conversions mainly on expected values and structural characteristics. Additional indirect matches, such as arithmetic computations and value transformations, are for future work. We also plan to semi-automatically construct domain ontologies used for expected values, automate application-dependent parameter tuning, and

test our approach in a broader set of real-world applications. As always, there is more work to do, but the results of our approach for both direct and indirect matching are encouraging, yielding over 90% in both recall and precision.

We formalized the output mapping between a target schema and source schema from our matching framework as a source-to-target mapping. By applying source-to-target mapping based on a predefined target schema, we propose TIQS, which provides solutions for five major issues: heterogeneity, scalability, continual infusion and change of local information sources, query processing complexity, and global-schema evolution in a unified approach to data integration. TIQS combines the advantages and avoids the limitations of both GAV and LAV. And it has polynomial-time query reformulation, and is easy to add or modify information sources. In summary, TIQS increases both scalability and usability as compared to previously proposed data-integration approaches.

References

[BCV99] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, March 1999.

[BE03] J. Biskup and D.W. Embley. Extracting information from heterogeneous information sources using ontologically specified target views. *Information Systems*, 28(3):169–212, 2003.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, Menlo Park, California, 1999.

[CA99] S. Castano and V. De Antonellis. ARTEMIS: Analysis and reconciliation tool environment for multiple information sources. In *Proceedings of the Convegno Nazionale Sistemi di Basi di Dati Evolute (SEBD'99)*, pages 341–356, Como, Italy, June 1999.

[CAFP98] S. Castano, V. De Antonellis, M.G. Fugini, and B Pernici. Conceptual schema analysis: Techniques and applications. *ACM Transactions on Database Systems*, 23(3):286–333, September 1998.

[CCGL02] A. Cali, D. Calvanese, G. De Giacomo, and M. Lenzerini. On the expressive power of data integration systems. In *Proceedings of 21st International Conference on Conceptual Modeling (ER2002)*, pages 338–350, Tampere, Finland, October 2002.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, October 1994.

[CLL01] D. Calvanese, D. Lembo, and M. Lenzerini. Survey on methods for query rewriting and query answering using views. Technical report, University of Rome, Roma, Italy, April 2001.

[DDH01] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 509–520, Santa Barbara, California, May 2001.

[ECJ+99] D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, and R.D. Smith. Conceptual-model-based data extraction from multiple-record Web pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[EJX01] D.W. Embley, D. Jackman, and Li Xu. Multifaceted exploitation of metadata for attribute match discovery in information integration. In *Proceedings of the International Workshop on Information Integration on the Web (WIIW'01)*, pages 110–117, Rio de Janeiro, Brazil, April 2001.

[EKW92] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach.* Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[Emb98] D.W. Embley. *Object Database Development: Concepts and Principles.* Addison-Wesley, Reading, Massachusetts, 1998.

[ETL02] D.W. Embley, C. Tao, and S.W. Liddle. Automatically extracting ontologically specified data from HTML tables with unknown structure. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER2002)*, pages 322–337, Tampere, Finland, October 2002.

[Fel98] C. Fellbaum. *WordNet: An Electronic Lexical Database.* MIT Press, Cambridge, Massachussets, 1998.

[FLM99] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI'99)*, pages 67–73, Orlando, Florida, 1999.

[GKD97] M.R. Genesereth, A.M. Keller, and O.M. Duschka. Infomaster: An information integration system. In *Proceedings of 1997 ACM SIGMOD International Conference on Management of Data*, pages 539–542, Tucson, Arizona, May 1997.

[Hal01] A.Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[HKL+01] J. Hu, R. Kashi, D. Lopresti, G. Nagy, and G. Wilfong. Why table ground-truthing is hard. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 129–133, Seattle, Washington, September 2001.

[LC00] W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Engineering*, 33(1):49–84, 2000.

[LRO96] A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 251–262, Mumbai (Bombay), India, September 1996.

[MBR01] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 49–58, Rome, Italy, September 2001.

[MHH00] R. Miller, L. Haas, and M.A. Hernandez. Schema mapping as query discovery. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB'00)*, pages 77–88, Cairo, Egypt, September 2000.

[MHH+01] R.J. Miller, M.A. Hernandez, L.M. Haas, L. Yan, C.T. Howard Ho, R. Fagin, and L. Popa. The clio project: Managing heterogeneity. *ACM SIGMOD Record*, 30(1):78–83, March 2001.

[Mil95] G.A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, November 1995.

[MWJ99] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proceedings of the Second International Conference on Information Fusion (FUSION 99)*, Sunnyvale, California, July 1999.

[MZ98] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB-98)*, pages 122–133, New York City, New York, August 1998.

[PTU00] L. Palopoli, G. Teracina, and D. Ursino. The system DIKE: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *Proceedings of ADBIS-DASFAA 2000*, pages 108–117, 2000.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[RB01] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[SC90] L. Tanca S. Ceri, G. Gottlob, editor. *Logic Programming and Databases*. Springer-Verlag, Berlin Heidelberg, 1990.

[Ull88] J.D. Ullman. *Pinciples of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press, New York, 1988.

[Ull97] J.D. Ullman. Information integration using logical views. In F.N. Afrati and P. Kolaitis, editors, *Proceedings of the 6th International Conference on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40, Delphi, Greece, January 1997.

[XE03] L. Xu and D.W. Embley. Discovering direct and indirect matches for schema elements. In *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA 2003)*, Kyoto, Japan, March 2003.