A Composite Approach to Automating Direct and Indirect Schema Mappings [*]

Li Xu
Department of Computer Science
University of Arizona South
lxu@email.arizona.edu

David W. Embley
Department of Computer Science
Brigham Young University
embley@cs.byu.edu

Abstract

Automating schema mapping is challenging. Previous approaches to automating schema mapping focus mainly 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 composite approach for generating a source-to-target mapping that contains both direct and 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 needed to compute indirect matches. Experiments we have conducted over several real-world application domains show encouraging results, yielding about 90% precision and recall measures for both direct and indirect matches.

1 Introduction

In this paper, we focus on the long-standing and challenging problem of automating schema mapping [RB01]. Schema mapping is a key operation for many applications including data integration, schema integration, message mapping in E-commerce, and semantic query processing [RB01]. Schema mapping takes two schemas as input and produces as output a semantic correspondence between the 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. In its simplest form, the semantic correspondence is a set of *direct element matches* each of which binds a source schema element to a target schema element if the two schema elements are semantically equivalent. To date, most research [BCV99, BM01, BM02, CAdV01, DDH01, DMD$^+$03, DR02, EJX01, HC03, KN03, LC00, MBR01, MGMR02, MZ98, PTU00] has focused on computing direct element matches. Such simplicity, however, is rarely sufficient, and researchers have thus proposed the use of queries over source schemas to form virtual elements to bind with target schema elements [BE03, DLD$^+$04, DMD$^+$03, MHH00]. In this more complicated form, the semantic correspondence

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

is a set of *indirect element matches* each of which binds a virtual source schema element to a target schema element through appropriate *manipulation operations* over a source schema.

We assume that all source and target schemas are described using rooted conceptual-model graphs (a conceptual generalization of XML). Element nodes either have associated data values or associated object identifiers. 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 domain, we construct a lightweight domain ontology [ECJ$^+$99], which declares regular-expression recognizers for concepts as well as relationships among concepts. We use the regular-expression recognizers to discover both direct and indirect matches between two arbitrary schemas. Based on the graph structure and these augmentations, we exploit a broad set of techniques together in a composite approach to settle direct and indirect element matches between a source schema and a target schema. As will be seen, regular-expression recognition and schema structure are the key ways to detect indirect element matches.

In this paper, we offer the following contributions: (1) a composite approach to automate identification of many indirect element matches between a source schema S and a target schema T as well as direct element matches, (2) an extension of relational algebra to express source-to-target mappings, and (3) experimental results of our implementation to show that our solution performs as well as (indeed better than) other approaches for direct matches and also performs exceptionally 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 needed algebra expressions for schema mappings. Section 3 describes a set of basic matching techniques to find potential element matches 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. In Section 4, we explain how to combine the confidences output from multiple basic matching techniques in our composite approach by applying a structure-matching technique. Section 5 presents a mapping algorithm to settle direct and indirect matches in a source-to-target mapping between S and T. Section 6 gives experimental results 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 this paper, we use a conceptual-modeling language OSM-L [Emb98, LEW00] to describe both source and target schemas, and an extension of the relational algebra to describe views over sources from which we create source-to-target mappings.

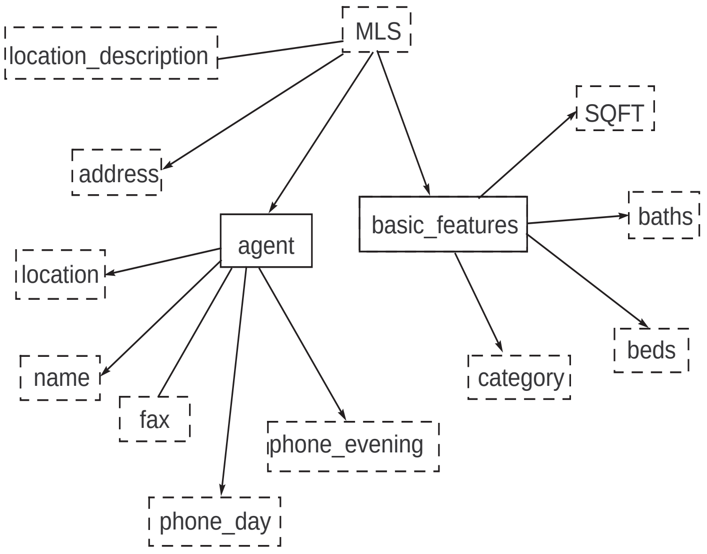
2.1 Target and Source Schemas

We use rooted graphs to represent both the target schema and the source schemas as conceptual-model specifications. Each conceptual schema has 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 let O_H denote the set of object sets and R_H denote the set of relationship sets in H. An object set contains either data values or object identifiers, which we respectively call a *lexical object set* or a *nonlexical object set*. A relationship set contains tuples of objects representing relationships connecting object sets. The root node is a designated object of primary interest. Figure 1, for example, shows two schema graphs. In a schema graph we denote lexical object sets as dashed boxes, nonlexical 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. For either a target or a source schema, we use an object/relationship-model instance to represent schema-level information in our approach for schema mapping.

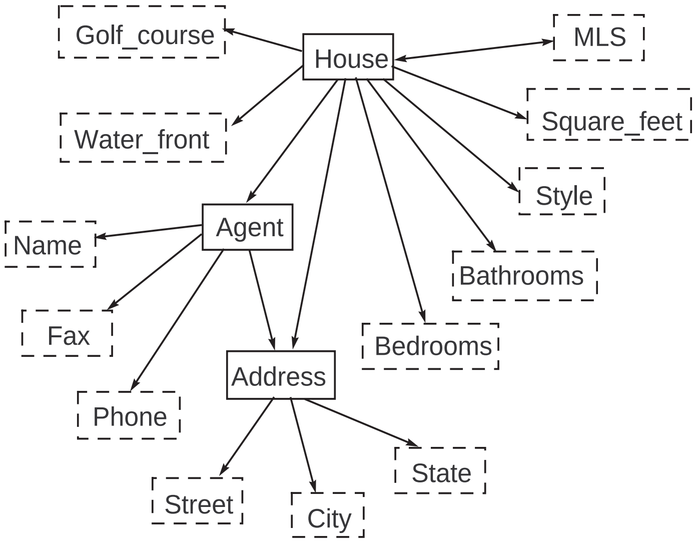
An optional component of a conceptual schema is a set of *data frames*, each of which describes the data of a lexical object set. A data frame is like a type which describes data instances, but can be much more expressive. A data-frame description can be as simple as a list of potential values for an object set and can be as complex as a regular-expression specification that represents values for the object set. For target and source schemas in this paper, data frames are lists of actual or sample values.

2.2 Source-to-Target Mappings

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. Our solution allows a variety of source derived data, including missing generalizations and specializations, merged and split values, transformation of attributes with Boolean



(a) Schema 1



(b) Schema 2

Figure 1: Conceptual-model graphs for Schema 1 and Schema 2

indicators into values and vice versa, lexicalization of object identifiers and vice versa, and schema paths as relationships. Therefore, our solution "extends" the source schema elements in Σ_H to include view schema elements, each of which we call a *virtual* object or relationship set. We let V_H denote the extension of Σ_H with derived, virtual object and relationship sets. In this paper, a schema element of H is a member of V_H and a virtual element of H is a member of $V_H - \Sigma_H$.

We consider a source-to-target mapping 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 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 that 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 ($\subseteq V_S$) 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 element through manipulation operations over a source schema. We denote a mapping element by $t \sim s \Leftarrow \theta_s(\Sigma_S)$, where $\theta_s(\Sigma_S)$ is a mapping expression that derives a source element s in V_S, and t is a target schema element in Σ_T. Note that the mapping expression may be degenerate so that $t \sim s$ is possible. Thus, as a formal definition, we say that the mapping element $t \sim s \Leftarrow \theta_s(\Sigma_S)$ is a *direct match* if the mapping expression is degenerate (i.e. $t \sim s$) and is otherwise an *indirect match*.

2.3 The Algebra for Source-to-Target Mappings

Each object and each relationship set (including derived object and relationship sets) in the target and source schemas are respectively single-attribute or multiple-attribute relations. Thus relational algebra applies directly to the object and relationship sets in a source or target schema. The standard 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 types of indirect matches we must face in creating virtual, derived object and relationship sets over source schemas.

- *Superset* and *Subset Values*. 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.

- *Lexicalization and Non-Lexicalization*. In Schema 2, the object identifiers in the object set $House$ represent the house-property objects. One object identifier in $House$ corresponds to one and only one MLS number in the object set MLS. Hence, the nonlexical object set $House$ in Schema 2 potentially matches with the object set MLS in Schema 1. Therefore, we need to lexicalize the object identifiers in $House$ with MLS numbers in Schema 2 if

Schema 1 is the target and compute virtual house-property object identifiers based on the MLS numbers in Schema 1 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 sets corresponding to the target relationship sets MLS—$basic\_features$ and $basic\_features$—$beds$; and if Schema 2 is the target, we need derive two virtual relationship sets corresponding to the target relationship sets $House$—MLS and $House$—$Bedrooms$.

Currently, we use the following operations over source schema elements to represent mapping expressions of indirect matches related to the above types we discussed. In the notation, a relation r has a set of attributes, which correspond to the names of lexical or nonlexical object sets; $attr(r)$ denotes the set of attributes in r; and $|r|$ denotes the number of tuples in r. For nonstandard 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 $\lambda_{(A_1,\ldots,A_n),A}r$ 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 Wymt Dr	Provo	Utah
h2	15 S 900 E	Provo	Utah
h3	75 Tiger Ln	Orem	Utah

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

House	Street	City	State	Address
h1	339 Wymt Dr	Provo	Utah	339 Wymt Dr, Provo, Utah
h2	15 S 900 E	Provo	Utah	15 S 900 E, Provo, Utah
h3	75 Tiger Ln	Orem	Utah	75 Tiger Ln, Orem, 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[1]. 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 to 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 Wymt Dr, Provo, Utah
h3	75 Tiger Ln, Orem, 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 Wymt Dr, Provo, Utah	339 Wymt Dr
h3	75 Tiger Ln, Orem, Utah	75 Tiger Ln

  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 Wymt Dr, Provo, Utah	Provo
h3	75 Tiger Ln, Orem, Utah	Orem

- *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 $attr(r) = \{House,\ Water\ Front\}$.

[1]A human expert is responsible to determine the routine R, which is domain dependent but is able to be applied across applications in the same domain.

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* $\reflectbox{$\beta$}$. The $\reflectbox{$\beta$}$ operator has the form $\reflectbox{$\beta$}^{Y,N}_{A,A'} r$, 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 $\reflectbox{$\beta$}^{\text{“}Yes\text{”},\text{“}No\text{”}}_{Lot\ Description, Water\ Front} r$ yields a new relation r', where $attr(r') = \{House,\ Water\ Front\}$.

House	Water Front
h1	Yes
h2	Yes
h3	No

Similarly, applying the operation $\reflectbox{$\beta$}^{\text{“}x\text{”},\text{“”}}_{Lot\ Description, Golf\ Course} r$ 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 three basic techniques to compare schema elements for schema mapping: (1) terminological relationships (e.g., synonyms and hypernyms), (2) data-value characteristics (e.g., string lengths and alphanumeric ratios), and (3) domain-specific, regular-expression matches (i.e. the appearance of expected strings). 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. For the third technique, we analyze data values based on domain ontologies to compute confidences and discover indirect matches as well as direct matches. The higher confidence values assigned by the techniques for a pair of schema elements, the more confident we have that the two elements are matchable.

3.1 Terminological Relationships

To match names of schema elements, 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, CAdV01]), but have given few, if any, details.

```
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 2: Generated WordNet rule by applying the C4.5 algorithm

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 a target schema element t. The C4.5-generated decision tree, however, was not intuitive.[2] 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 word 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 2 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 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

[2]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.

	MLS	*bath.*	*bed.*	*cat.*	*SQ.*	*loc._desc.*	*basic_feat.*	*agent*	*fax*	*ph._day*	*ph._even.*	*name*	*loc.*	*addr.*
House	0.11	0.12	0.12	0.11	0.11	0.12	0.11	0.11	0.11	0.11	0.11	0.98	0.12	0.11
Bathrooms	0.11	0.98	0.98	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
Bedrooms	0.11	0.98	0.98	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
MLS	0.98	0.11	0.11	0.11	0.11	0.43	0.11	0.11	0.11	0.11	0.11	0.43	0.11	0.43
Square_feet	0.11	0.11	0.11	0.11	0.98	0.11	0.11	0.11	0.43	0.27	0.27	0.12	0.11	0.11
Water_front	0.11	0.11	0.11	0.12	0.12	0.12	0.12	0.11	0.11	0.11	0.11	0.12	0.11	0.12
Golf_course	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
Address	0.43	0.11	0.11	0.11	0.11	0.98	0.12	0.11	0.11	0.11	0.11	0.12	0.11	0.98
Agent	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.98	0.11	0.11	0.11	0.11	0.11	0.11
Fax	0.11	0.11	0.11	0.11	0.43	0.11	0.11	0.11	0.98	0.67	0.67	0.11	0.11	0.11
Phone	0.11	0.11	0.11	0.11	0.43	0.11	0.11	0.11	0.67	0.98	0.98	0.98	0.11	0.11
Name	0.43	0.11	0.11	0.11	0.12	0.43	0.11	0.11	0.11	0.98	0.98	0.98	0.11	0.12
Street	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.98	0.11	0.43	0.11	0.11
State	0.11	0.11	0.11	0.11	0.12	0.11	0.11	0.11	0.11	0.11	0.11	0.12	0.11	0.11
City	0.11	0.11	0.11	0.11	0.67	0.11	0.11	0.11	0.11	0.11	0.11	0.43	0.11	0.11
Style	0.11	0.11	0.11	0.98	0.43	0.98	0.12	0.11	0.43	0.43	0.43	0.11	0.11	0.98

Figure 3: WordNet confidence-value matrix

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 2, 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 contain abbreviations, acronyms, or domain jargon, we rewrite them as ordinary natural-language words so that WordNet can recognize them.[3] If the names of both s and t are single-word tokens, the computation $conf_1(s,t)$ 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-$(x\text{-}y)/x$, 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 nouns, we use an injective, greedy match algorithm to locate the potential matching nouns between the name phrases of s and t. The algorithm takes the best matching pair of words and then eliminates the matched words from the name phrases in s and t before selecting the next best pair, and so forth. 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.

Assume that Schema 1 in Figure 1(a) is a source schema, and Schema 2 in Figure 1(b) is a target schema. Figure 3 shows a confidence-value matrix generated by the decision rule in Figure 2 for the target and source schemas. The schema elements along the top are source schema elements taken

[3]We currently do this rewriting manually, but it is possible to use dictionaries to do this semiautomatically.

from Schema 1.[4] The schema elements on the left are target schema elements taken from Schema 2. Observe, for example, that the confidence values $conf_1(agent, Agent)$, $conf_1(beds, Bedrooms)$, $conf_1(baths, Bathrooms)$, $conf_1(phone\_day, Phone)$, and $conf_1(phone\_evening, Phone)$ are high as they should be. Observe, however, that the two confidence values $conf_1(location\_description, Golf\_course)$ and $conf_1(location\_description, Water\_front)$ are low, even though "Golf_course" and "Water_front" around a house property are two kinds of "location_description"; and the confidences $conf_1(category, Style)$, $conf_1(location\_description, Style)$, $conf_1(address, Style)$ are high based on the WordNet hierarchical structure, even though the object sets do not semantically correspond with each other. As we shall see, however, other techniques can sort out and eliminate these anomalies.

3.2 Data-Value Characteristics

Previous work in [LC00] shows that characteristics among data values 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] calculated from sample data associated with object sets in a wide variety of applications, 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 2, we generate a confidence value, denoted $conf_2(s, t)$, for each element pair (s, t) of schema elements that has data values available.

Figure 4 shows a confidence-value matrix generated by 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. Note that in Figure 4 there are several nonlexical object sets whose values are object identifiers in Schema 1 and Schema 2. An *NA* in the matrix denotes that the object identifiers associated with either the source object set in a column or the target object set in a row are not applicable for value analysis. Observe that the confidence values such as $conf_2(beds, Bedrooms)$, $conf_2(baths, Bathrooms)$, $conf_2(phone\_day, Phone)$, and $conf_2(fax, Fax)$ are high, as expected. Observe, however, several high confidence values produced are not correct. For example, Fax in the target and *phone_day* in the source tend to look alike according to the value characteristics

[4]In order to fit the table in the page, we use abbreviations for schema-element names in the source.

	MLS	*bath.*	*bed.*	*cat.*	*SQ.*	*loc._desc.*	*basic_feat.*	*agent*	*fax*	*ph._day*	*ph._even.*	*name*	*loc.*	*addr.*
House	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Bathrooms	0.33	0.86	0.33	0.08	0.86	0.08	*NA*	*NA*	0.33	0.33	0.33	0.08	0.33	0.33
Bedrooms	0.33	0.31	0.86	0.08	0.33	0.08	*NA*	*NA*	0.33	0.31	0.31	0.08	0.33	0.33
MLS	0.86	0.67	0.31	0.08	0.33	0.08	*NA*	*NA*	0.33	0.67	0.67	0.08	0.33	0.33
Square_feet	0.33	0.86	0.33	0.08	0.86	0.08	*NA*	*NA*	0.91	0.33	0.33	0.08	0.33	0.33
Water_front	0.08	0.05	0.05	0.33	0.05	0.33	*NA*	*NA*	0.05	0.08	0.08	0.33	0.08	0.08
Golf_course	0.08	0.05	0.05	0.33	0.05	0.33	*NA*	*NA*	0.05	0.08	0.08	0.33	0.08	0.08
Address	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Agent	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Fax	0.91	0.33	0.33	0.08	0.91	0.08	*NA*	*NA*	0.86	0.86	0.86	0.08	0.33	0.33
Phone	0.31	0.33	0.33	0.08	0.91	0.08	*NA*	*NA*	0.86	0.86	0.86	0.08	0.33	0.33
Name	0.08	0.08	0.08	0.86	0.08	0.86	*NA*	*NA*	0.08	0.1	0.1	0.86	0.1	0.1
Street	0.91	0.33	0.33	0.91	0.33	0.1	*NA*	*NA*	0.86	0.86	0.86	0.91	0.31	0.91
State	0.08	0.05	0.05	0.33	0.05	0.33	*NA*	*NA*	0.05	0.08	0.08	0.33	0.08	0.08
City	0.08	0.08	0.08	0.91	0.08	0.33	*NA*	*NA*	0.08	0.1	0.1	0.33	0.08	0.08
Style	0.91	0.08	0.08	0.67	0.08	0.67	*NA*	*NA*	0.08	0.08	0.08	0.86	0.08	0.08

Figure 4: Value-characteristics confidence-value matrix

measured, an incorrect match 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 high similar value characteristics with style values in *Style* of the target schema. 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

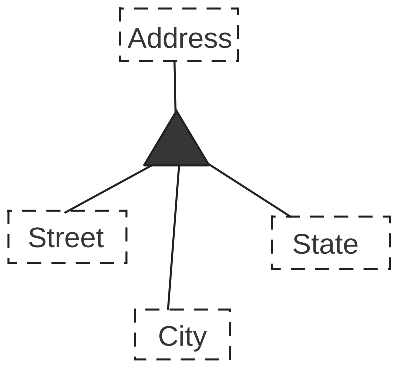
Whether expected values appear in a set of data provides yet another a clue to which elements match. For a specific application domain, we can specify lightweight a domain ontology [ECJ+99], which includes a set of concepts and relationships among the concepts, and associates with each concept a set 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 domain 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. Figure 5 shows the regular expressions using the Perl syntax we specified for two concepts in a lightweight domain ontology for a real-estate domain.[5]

We declare the concepts and relationship sets in our lightweight domain ontology independently of any target and source schemas. We call the ontology lightweight for three reasons. (1) We neither require nor expect that the knowledge declared in the domain ontology to be complete for the domain. (2) The objective of the regular expressions declaring expected values for domain concepts

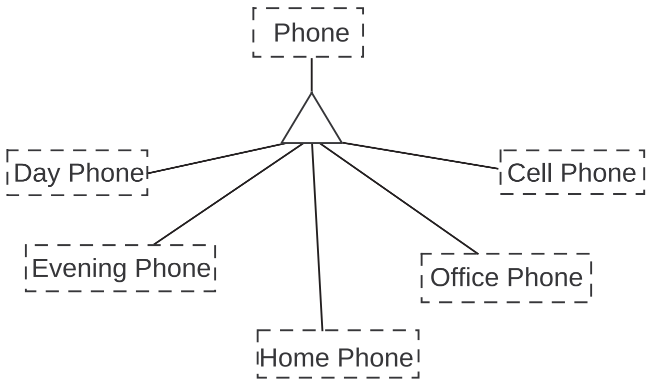
[5][Hew00] provided a user-friendly tool to create the regular-expression specifications.

```
View matches [15] case insensitive
    constant
        { extract "\bmountain\sview\b"; },
        { extract "\bocean\sview\b"; },
        { extract "\briver\sview\b"; },
        { extract "\bbay\sview\b"; },
        { extract "\bharbor\sview\b"; },
        { extract "\bwater\sview\b"; },
        { extract "\bpanoramic\sview\b"; },
        { extract "\bcity\slight(s)?\b"; },
        { extract "\bcity\sview\b"; },
        { extract "\bvalley\sview\b"; },
        { extract "\bgardon\sview\b"; },
        { extract "\bpool\sview\b"; },
        { extract "\bgolf\s*course\sview\b"; },
        { extract "\bcoastline\sview\b"; },
            ...
        { extract "\bgreenbelt\sview\b"; };
    keyword
        "\bview(s)?\b";
End;
Phone matches [15] case insensitive
    constant
        { extract "\b\d{3}-\d{4}\b"; }, – nnn-nnnn
        { extract "\b\(\d{3}\)\s*\d{3}-\d{4}\b"; }, – (nnn) nnn-nnnn
        { extract "\b\d{3}-\d{3}-\d{4}\b"; }, – nnn-nnn-nnnn
        { extract "\b\d{3}\\\d{3}-\d{4}\b"; }, –nnn\nnn-nnnn
        { extract "\b1-\d{3}-\d{3}-\d{4}\b"; }; – 1-nnn-nnn-nnnn
    Keyword
        "\bcall\b;
End;
```

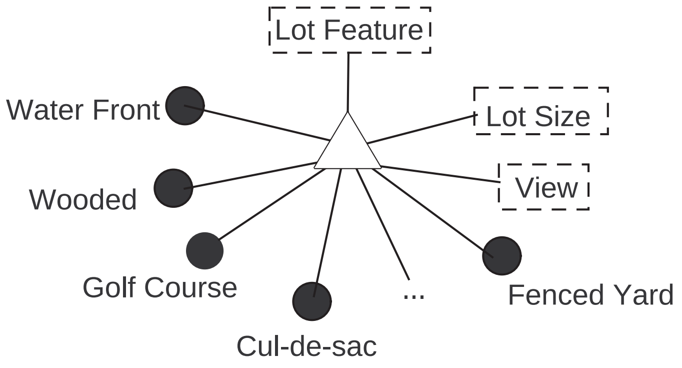
Figure 5: Example of regular expressions specified for concepts in the lightweight domain ontology



(a) Address



(b) Phone



(c) Lot Feature

Figure 6: Application domain ontology (partial)

is to discover corresponding concepts, not to extract items of interest [ECJ+99]. Thus, they need not be as exact or as comprehensive as regular expressions for data-extraction ontologies [ECJ+99]. (3) Because they are usually small and because we can often reuse many regular expressions for data items such as date, time, and currency which cross domains, it often only takes on the order of a day or so [6] to construct a new domain ontology.

Figure 6 shows three components in our real-estate domain ontology, which we used to automate

[6]We asked 24 students who have taken CS652, the extraction and integration course at BYU, to report the number of hours it took them to create extraction ontologies for a new domain. Projects included, for example, pharmaceutical drugs, jewelry, TV's, and camp grounds, etc. All reported taking somewhere in the neighborhood of about two dozen hours or less.

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*, and *Street*;[7] a phone component specifying *Phone* as a possible superset of *Day Phone*, *Evening Phone*, *Home Phone*, *Office Phone*, and *Cell Phone*;[8] and a lot-feature component specifying *Lot Feature* as a possible superset of *View* and *Lot Size* values and individual values *Water Front*, *Golf Course*, etc.[9] Behind a dashed box (or individual value), a regular-expression recognizer [ECJ+99] describes the expected-data values for a potential domain concept. The ontology explicitly declares that (1) the expected values in *Address* match with a concatenation of the expected values for *Street*, *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, the set of *Lot Size* values, the singleton-sets including *Water Front*, *Golf Course*, *Wooded*, *Fenced Yard*, $Cul-de-sac$, etc.

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 introduce the idea with examples and then more formally explain how this works in general.)

1. *Merged* and *Split Values.* Based on the *Address* declared in the ontology in Figure 6, 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 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 the *Composition* operation over the source to merge values in *Street*, *City*, and

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

State to directly match with *address* in the target.[10]

2. *Superset* and *Subset Values.* Based on the specification of the regular expression for *Phone*, the schema elements *phone_day* and *phone_evening* in Schema 1 of Figure 1(a) match with the 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 its set of expected values 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 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* operator to sort out which phones belong in which specialization (if not, a human expert may not be able to sort these out either).

3. *Object-Set 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-object-set 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 object-set 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 a domain 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

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

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 h_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 h_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 object-set names, we distribute the object-set names over all the Boolean fields that have "Yes" values.

We decide if s matches with t directly or indirectly by comparing C_s and C_t if the hit ratios h_s and h_t are above an accepted threshold. 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 object-set 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 object-set 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* operators for $(s,\ t)$, 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 $(cos(v_s, v_t) \times (h_s + h_t)/2)$.

Figure 7 shows the matrix containing confidence values computed based on expected-data values using Schema 1 in Figure 1(a) as a source schema and Schema 2 in Figure 1(b) as a target schema. Observe that the technique correctly identifies the indirect matches between *location_description* in

	MLS	*bath.*	*bed.*	*cat.*	*SQ.*	*loc._desc.*	*basic_feat.*	*agent*	*fax*	*ph._day*	*ph._even.*	*name*	*loc.*	*addr.*
House	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Bathrooms	0.0	*NA*	*NA*	*NA*	0.0	0.0	*NA*	*NA*	0.0	0.0	0.0	0.0	0.0	0.0
Bedrooms	0.0	*NA*	*NA*	*NA*	0.0	0.0	*NA*	*NA*	0.0	0.0	0.0	0.0	0.0	0.0
MLS	1.0	0.0	0.0	0.0	0.0	0.0	*NA*	*NA*	0.0	0.0	0.0	0.0	0.0	0.0
Square_feet	0.0	0.0	0.0	0.0	1.0	0.0	*NA*	*NA*	0.0	0.0	0.0	0.0	0.0	0.0
Water_front	0.0	0.0	0.0	0.0	0.0	1.0	*NA*	*NA*	0.0	0.0	0.0	0.0	0.0	0.0
Golf_course	0.0	0.0	0.0	0.0	0.0	1.0	*NA*	*NA*	0.0	0.0	0.0	0.0	0.0	0.0
Address	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Agent	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Fax	0.0	0.0	0.0	0.0	0.0	0.0	*NA*	*NA*	1.0	0.0	0.0	0.0	0.0	0.0
Phone	0.0	0.0	0.0	0.0	0.0	0.0	*NA*	*NA*	0.0	1.0	1.0	0.0	0.0	0.0
Name	0.0	0.0	0.0	0.0	0.0	0.0	*NA*	*NA*	0.0	0.0	0.0	1.0	0.0	0.0
Street	0.0	0.0	0.0	0.0	0.0	0.0	*NA*	*NA*	0.0	0.0	0.0	0.0	1.0	1.0
State	0.0	0.0	0.0	0.0	0.0	0.0	*NA*	*NA*	0.0	0.0	0.0	0.0	1.0	1.0
City	0.0	0.0	0.0	0.0	0.0	0.0	*NA*	*NA*	0.0	0.0	0.0	0.0	1.0	1.0
Style	0.0	0.0	0.0	0.0	0.0	0.0	*NA*	*NA*	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7: Expected-data-values confidence-value matrix

the source and *Golf_course* and *Water_front* in the target, between *phone_day* and *phone_evening* in the source and *Phone* in the target, and between *address* and *location* in the source and *Street*, *City*, and *State* in the target. Once again note that the object identifiers associated with nonlexical object sets in both target and source schemas are inapplicable for the expected-data-values analysis. Furthermore, for this example, we did not include the specifications for expected-data values of "bedrooms" and "bathrooms" in our lightweight ontology. The values for *Bedrooms* and *Bathrooms* in the target and the values for *beds* and *baths* in the source do not match any concept in the domain ontology. If one set of data values corresponds to the expected-data values specified for a concept and another set of data values does not correspond to any concept in the ontology, the confidence is 0.0. For example, the confidence $conf_3(baths, Phone)$ is 0.0 because the values for *Phone* in the target correspond to the concept *Phone* in the ontology, but the values for *baths* in the source do not. If neither values of a pair corresponds to any concept specification in the ontology,[11] the entry is *NA*. For example, the *NA* for the pair (*baths*, *Bathrooms*) denotes that the data values for neither *baths* in the source nor *Bathrooms* in the target match any concept in the lightweight real-estate domain ontology. If the domain ontology are not complete with respect to an application, our approach needs other matching techniques to discover matches that are not discovered through comparing expected-data values.

4 Structure Matching

The three matching techniques including the terminological relationships in Section 3.1, the value characteristics in Section 3.2, and the expected-data values in Section 3.3 compare only object

[11]We are not able to compare the expected-data values without the help of the domain ontology.

sets between a source schema and a target schema. In addition to object-set matches, structure matching applies to 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. For example, assume that we let Schema 1 of Figure 1(a) be a target schema and Schema 2 of Figure 1(b) be a source schema. Based on the terminological relationships, value characteristics, and expected-data values, we obtain object-set matches such as the match between MLS and MLS and the match between *Bedrooms* and *beds*. Without relationship-set matches, however, it may be impossible to correctly answer a user query such as "Find houses with 4 bedrooms" because the query requires the system to match the semantically equivalent relationships in the source with the relationships between MLS and *beds* in the target.

In a source-to-target mapping, a mapping element $t \sim s \Leftarrow \theta_s(\Sigma_S)$ 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 over the source, is semantically equivalent to a target element t, we can access the data of s through the target element t. Intuitively, a source-to-target mapping between a source schema and a target schema describes all the needed access paths to retrieve data facts from the source for matching target elements.

When comparing structural properties of a target schema and a source schema, we apply a top-down strategy. At the top level, we compute semantic correspondences between abstract components of the target and source schemas. Each of the components for all schemas is composed of a set of object sets and relationship sets among the object sets. The structure-matching technique determines the composition of abstract components 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 object and relationship sets.

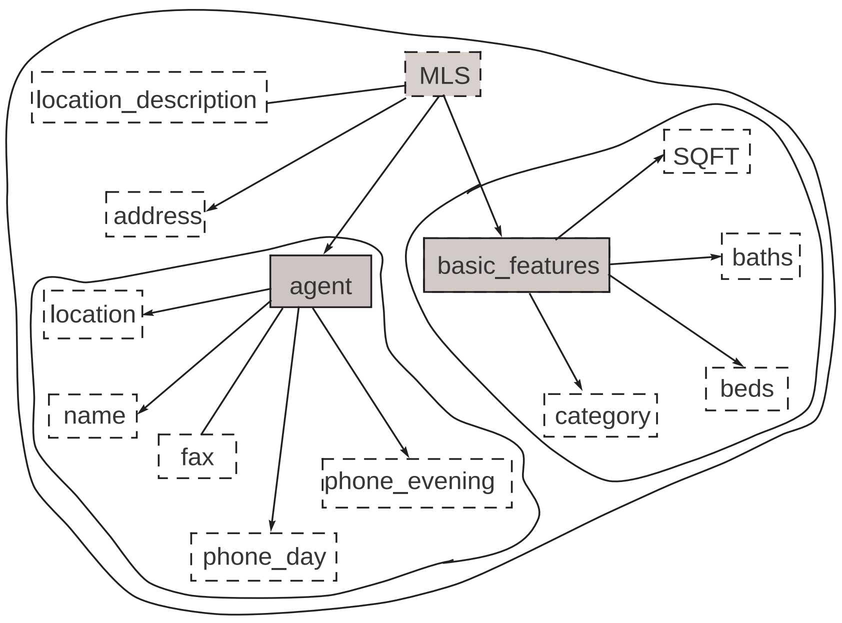
Abstract components in target and source schemas make use of equivalence classes of object sets [Emb98]. Based on the relationship-set constraints in a conceptual schema H, we partition the object sets of H into equivalence classes in which the objects of the object sets of an equivalence class are in a one-to-one correspondence. Let X and Y be subsets of the object sets in H, and let F be a set of functional dependencies over H as denoted by the functional edges in H. The two subsets 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. An equivalence class is *trivial* if it only contains a single object set. Otherwise, the equivalence class is *nontrivial.* In Schema 2 of Figure 1(b), one non-trivial equivalence class is $\{House,\ MLS\}$. The other equivalence classes in Schema 2 and all the equivalence classes in Schema 1 are trivial.

We further analyze equivalence classes in a conceptual schema H and divide them into a set of representative equivalence classes, which we denote as E_H^R, and a set of non-representative equivalence classes. Intuitively, the set of representative equivalence classes of a conceptual schema H consists of those equivalence classes that are most important and informative for H. Formally, $e \in E_H^R$ is a *representative equivalence class* if and only if (1) the equivalence class e is nontrivial, or (2) the object set in e determines other set of object sets in H. In Figure 1, the representative equivalence classes in Schema 1 of Figure 1(a) are $\{MLS\}$, $\{agent\}$, and $\{basic\_features\}$ and the representative equivalence classes in Schema 2 of Figure 1(b) are $\{House,\ MLS\}$, $\{Agent\}$, and $\{Address\}$.

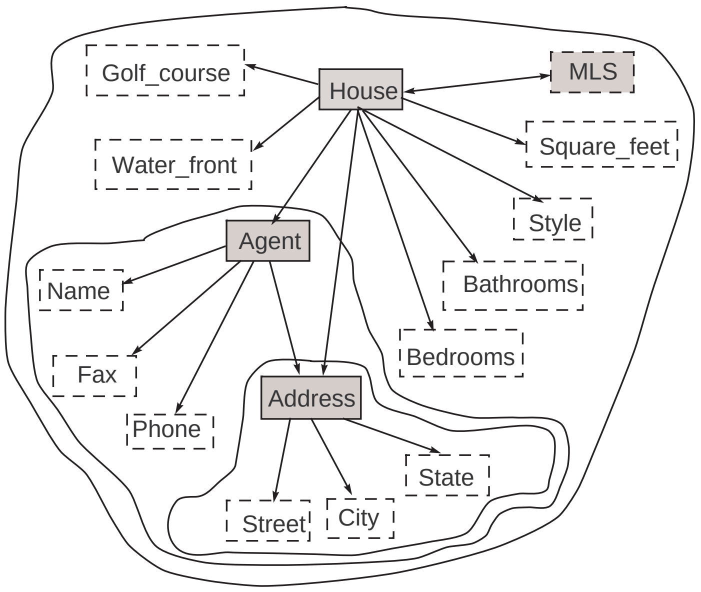
In addition to representative equivalence classes, the structure-matching technique makes use of one other notion, "context" of equivalence classes. By taking relationship sets around a representative equivalence class e into account, we cluster a set of object sets and relationship sets with a representative equivalence class as an abstract component in the schema. We call this component the *context* for the representative equivalence class e, which we denote as $Cont_e$. The context $Cont_e$ for a representative equivalence class e consists a set of object sets, which we denote as $Cont_e^O$, and a set of relationship sets, which we denote as $Cont_e^R$, among the object sets in $Cont_e^O$.

We create the context in two phases. In the first phase, we construct a context beginning with each representative equivalence class e as follows.[12] (1) Include all object sets in the functional closure h^+ of any element h of e. (2) Include all object sets adjacent to object sets in h^+ except object

[12]Context can be defined in different ways. Believing that all object sets immediately adjacent to a schema element x are relevant, we chose to include them in our context. In addition, similar to the idea of giving higher weights to functionally dependent information in measuring closeness [CAFP98], we also chose to include object sets functionally dependent on x as well as their immediately adjacent object sets (unless we encounter a representative equivalence class).



(a) Schema 1



(b) Schema 2

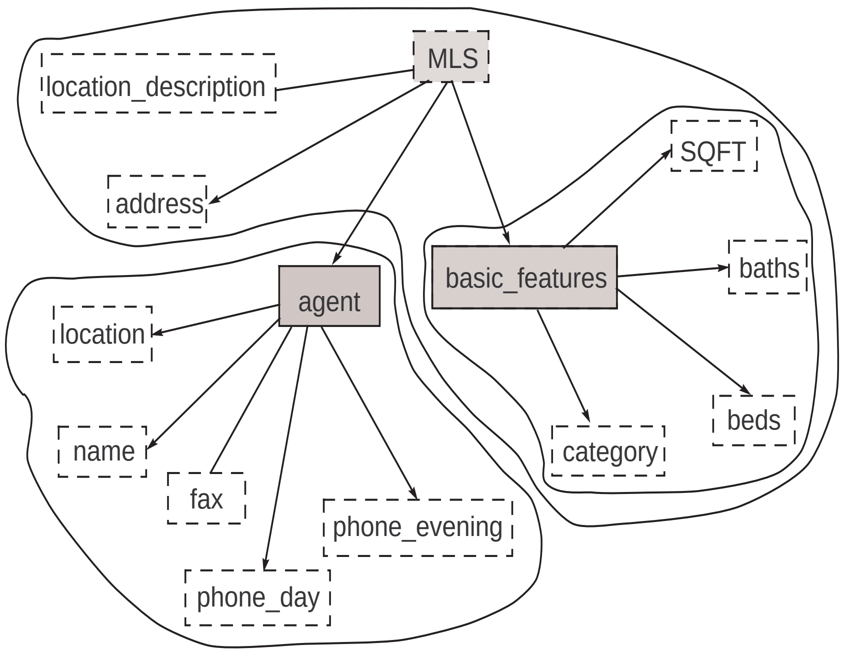
Figure 8: Context construction for Schema 1 and Schema 2 after the first phase

```
Input: a representative equivalence class $e$ in a schema $H$.
Output: a context $Cont_e$ of $e$.
    Include all the object sets in $e$ into a set $Cont_e^O$
    for each equivalence class $e'$ connected by $e$
        if $e'$ is non-representative
            Include the object set in $e'$ into $Cont_e^O$
        else if $e'$ is not compatible and $e \rightarrow e'$
            Include the object sets in the context of $e'$ into $Cont_e^O$
    Collect relationship sets among $Cont_e^O$ in $H$ into a set $Cont_e^R$
    Output the union of $Cont_e^O$ and $Cont_e^R$ as $Cont_e$
```

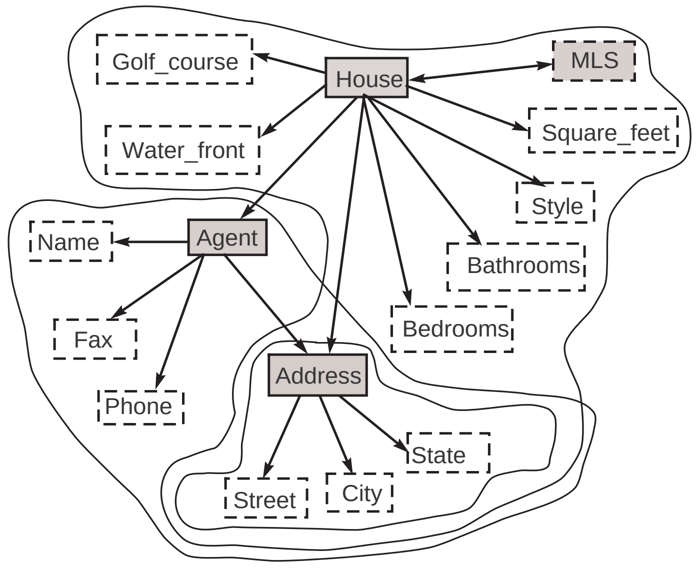
Figure 9: Context construction for representative equivalence classes

sets that are members of some other representative equivalence class. (3) Include all relationship sets that connect the included object sets. Figure 8 shows the construction after the first phase. In Figure 8, the representative equivalence-class elements are shaded and each enclosed area includes the object and relationship sets of the context for a representative equivalence class after the first phase. Consider, as an example, the first-phase construction of the context for MLS in Figure 8(a). The representative equivalence class e is $\{MLS\}$. The closure MLS^+ includes all the object sets in Figure 1(a) except *location_description* and fax. Since both *location_description* and fax, however, are adjacent to object sets included in MLS^+, we also include both *location_description* and fax in the first phase of construction of the context for MLS.

After the first phase of context construction for representative equivalence classes in target and source schemas, we compare the representative equivalence classes as well as their contexts. Between two schemas, we decide on a set of *compatible* representative equivalence classes, each of which is determined according to available confidence values for two compared representative equivalence classes as well as their respective contexts. In Figure 8, we determine that the representative equivalence class $\{House,\ MLS\}$ is compatible with the representative equivalence class $\{MLS\}$ and likewise that $\{Agent\}$ is compatible with $\{agent\}$ because both the object sets and the contexts of the equivalence classes are largely matchable. Then, in the second phase of context construction for representative equivalence classes, we use the algorithm in Figure 9 to further construct the contexts for compatible representative equivalence classes in the target and source schemas. Figure 10 shows the new contexts for representative equivalence classes of schemas in Figure 1. Note that the contexts for the compatible representative equivalence classes $\{MLS\}$ and $\{House,\ MLS\}$ are smaller than their contexts in Figure 8 after the first phase. Essentially, the second phase separates



(a) Schema 1



(b) Schema 2

Figure 10: Context construction for Schema 1 and Schema 2 after the second phase

the contexts of the elements into different contexts. This reduces the context scopes for representative equivalence classes and thus limits the search of finer-level correspondences to appropriate, smaller search spaces.

The structure-matching technique discovers object-set matches as well as most relationship-set matches between contexts of compatible representative equivalence classes. We base our approach to structure matching for both direct and indirect matches on four intuitive ideas, which we illustrate using Schemas 1 and 2 in Figure 10.

1. *Nonlexical object sets.* Two nonlexical object sets match if their element names match and they are in contexts of compatible equivalent classes—the lexical object sets around them describe matchable data in the two schemas. A nonlexical object set has only object identifiers in a target or source schema. The object identifiers themselves do not describe the objects in the nonlexical object set. Instead, the values of object sets around the nonlexical object set describe nonlexical objects. The context analysis provides a limited scope for selecting the lexical object sets around the nonlexical object sets. In Figure 10, both *agent* in Schema 1 and *Agent* in Schema 2 represent the agent object for a house. The confidence $conf_1(agent,\ Agent)$, computed based on terminological relationships between the two names, declares that *Agent* of Schema 2 matches *agent* of Schema 1 in Figure 10. The data associated with the adjacent *location*, *name*, fax, *phone_day*, and *phone_evening* together describe objects of *agent* in Schema 1. Similarly, the data associated with the adjacent $Name$, Fax, and $Phone$ as well as the adjacent *Street*, *City*, *State* around the object *Address* together describe objects of *Agent* in Schema 2. By taking the adjacent lexical object sets into account, *agent* of Schema 1 does match with *Agent* of Schema 2.

2. *Equivalence classes.* The equivalence class $\{House,\ MLS\}$ of Schema 2 matches the equivalence class $\{MLS\}$ of Schema 1. With respect to the equivalence classes, we can identify a match between the two MLS object sets and we can see the matchable lexical object sets that are closely related as already discussed in the above paragraph for nonlexical object-set matches. Assuming that Schema 1 is a source schema and Schema 2 is a target schema, we can create a virtual nonlexical object set $House'$ whose object identifiers are in a one-to-one correspondence with the values for MLS in Schema 1. The virtual nonlexical object set

$House'$ matches with $House$ in Schema 2. Assuming, on the other hand, that Schema 2 is a source schema and Schema 1 is a target schema, we can simply match the MLS object sets directly.

3. *Lexical object sets.* Closely related sets of objects and values supply additional constraints for matching lexical object sets. In Figure 10(a), $address$ and *location* denote house and agent locations respectively. Based on an analysis of the contexts for the two schemas, Figure 10(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 10(b) for *location* and $address$ by considering contexts. In Figure 10(b), the $Address$ objects are in both the context for $\{House,\ MLS\}$ and for {*Agent*}. With the compatibility between $\{House,\ MLS\}$ and $\{MLS\}$, given that the values for $Street$, *City*, and $State$ describe addresses in Figure 10(b), we can decide that there exists an indirect match between the addresses in the context of $\{MLS\}$ in Figure 10(a) and the addresses in the context of $\{House,\ MLS\}$ in Figure 10(b). Similarly, we can decide the other indirect match between the addresses in the context of {*agent*} in Figure 10(a) and the addresses in the context of {*Agent*} in Figure 10(b).

4. *Relationship sets.* In addition to matching object sets, we also match relationship sets, some of which may be virtual. Intuitively, *agent—name* in Figure 10(a) matches *Agent—Name* in Figure 10(b) and *MLS—agent* in Figure 10(a) matches the virtual relationship set $\pi_{MLS,Agent}(MLS\text{—}House \bowtie House\text{—}Agent)$ in Figure 10(b). However, *agent—fax* in Figure 10(a) does not immediately match *Agent—Fax* in Figure 10(b) because *agent—fax* is many-many while *Agent—Fax* is functional. A source relationship set r_s, which could be a virtual element, matches with a target relationship set r_t if and only if the two relationship sets r_s and r_t are isomorphic, i.e. a mapping function f_r from r_s to r_t exists such that there is an object set $f_r(o_{r_s})$ with constraint[13] c connected by r_t if and only if there is an object set o_{r_s} with constraint c connected by r_s. Thus the requirement for relationship-set matches falls into two categories: (1) type requirements to satisfy node matching, and (2) constraint requirements to satisfy edge isomorphism. The type requirement between two nodes o_{r_s} and

[13]Here, the constraint c is the functional constraint, if it exists, that may be declared in the conceptual model instance as explained in Section 2.

o_{r_t} is satisfied if and only if there exists an object-set match $o_{r_t} \sim o_{r_s} \Leftarrow \theta_{o_{r_s}}(\Sigma_S)$. To check constraint compatibility between two connections, [BE03] proposed four cases, which guide users' involvements for schema mapping operations while translating source data into the target. We adopt these same four cases for our work as follows. (1) The constraints on r_s and r_t are equivalent. Since this case satisfies the isomorphism constraints, nothing further need be done. (2) The constraints of r_s imply the constraints of r_t but not vice versa. In this case, since the relationships for r_s already necessarily satisfy the constraints of the target relationship set r_t, there is nothing further we need to do. For example, since the functional constraint of *Agent—Fax* is stronger than the many-many constraint of *agent—fax*, if Figure 10(b) is the source and Figure 10(a) is the target, we can make the match with nothing further to do. (3) The constraints of r_t imply the constraints of r_s but not vice versa. In this case, we need to further transform r_s using *Selection* to restrict r_s to r'_s so that r'_s contains only relationships that satisfy the constraints of r_t. As a default, we can select relationships from r_s in an arbitrary order and keep only those that satisfy the constraints of r_t; otherwise a DBA must specify the selection criteria.[14] Here, for example, since the functional constraint of *Agent—Fax* is stronger than the many-many constraint of *agent—fax*, if Figure 10(a) is the source and Figure 10(b) is the target, we can only make the match if we say which of the (presumably) several fax machines of the agent to select. (4) Neither the constraints of r_t imply the constraints of r_s nor vice versa. This is a combination of (2) and (3) and thus, since there is nothing to do for (2), we can transform r_s as explained in (3).

Since our approach for schema mapping allows derived data in source schemas, an 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 have exponential time complexity. To avoid generating a large number of views over a source schema, we restrict the search space for view generation. Our structure matching technique first discovers semantic correspondences between object sets. Then, with the guide of object-set correspondences, it discovers relationship-set matches. Intuitively, we want to use the type requirements for relationship-set matches to trigger derivations of virtual elements over the source within a subset of a context, where the subset consists of a set of object sets and a

[14]See [BE03] for a detailed explanation of possibilities.

set of relationship sets among the object sets. The selection of the object sets is based on object-set matches and context restrictions. For example, let Schema 1 in Figure 10(a) be a target schema and Schema 2 in Figure 10(b) be a source schema, and assume that we discover that the objects in *Agent* in the context of {*Agent*} corresponds with the objects in *agent* in the context of {*agent*} and that the values for *Street*, *City*, and *State* of objects *Address* in the context of {*Agent*} in Schema 2 semantically corresponds to *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—City*, and *Address—State* in the source in order to form a virtual relationship set that matches *agent—location* in the target.

5 Mapping 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. To illustrate our algorithm, we use our running example in Figure 1, and 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 nonlexical object sets, the algorithm computes a confidence value, $conf(s,t)$, as a combination of the output confidence values of the three nonstructural matching techniques as described in Section 3. We compute $conf(s,t)$ using the following formula:

$$conf(s,t) = \begin{cases} conf_1(s,t) \text{ , if } s \text{ and } t \text{ are nonlexical object sets} \\ 1.0 \text{ , if } conf_3(s,t) = 1.0 \text{ and } s \text{ and } t \text{ are lexical object sets} \\ w_s(conf_1(s,t)) + w_v(conf^v(s,t)) \text{ , otherwise} \end{cases}$$

In this formula, w_s and w_v are experimentally determined weights.[15] When the confidence value $conf_3(s,t) = 1.0$, we let $conf_3$ dominate and assign $conf(s,t)$ as 1.0. 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

[15]The two parameters w_s, which weights schema element names, and w_v, which weights schema element values, are domain 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.

	MLS	*bath.*	*bed.*	*cat.*	*SQ.*	*loc._desc.*	*basic_feat.*	*agent*	*fax*	*ph._day*	*ph._even.*	*name*	*loc.*	*addr.*
House	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	0.11	0.11	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Bathrooms	0.14	0.92	0.66	0.09	0.27	0.07	*NA*	*NA*	0.14	0.14	0.14	0.07	0.14	0.14
Bedrooms	0.14	0.65	0.92	0.09	0.14	0.07	*NA*	*NA*	0.14	0.13	0.13	0.07	0.14	0.14
MLS	1.0	0.22	0.13	0.07	0.14	0.23	*NA*	*NA*	0.14	0.22	0.22	0.23	0.14	0.3
$Square_Feet$	0.14	0.27	0.14	0.07	1.0	0.07	*NA*	*NA*	0.44	0.22	0.22	0.08	0.14	0.14
$Water_Front$	0.07	0.07	0.07	0.14	0.07	1.0	*NA*	*NA*	0.07	0.07	0.07	0.14	0.07	0.08
$Golf_course$	0.07	0.07	0.07	0.14	0.07	1.0	*NA*	*NA*	0.07	0.07	0.07	0.14	0.07	0.07
Address	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	0.12	0.11	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Agent	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	0.11	0.98	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
Fax	0.28	0.14	0.14	0.07	0.44	0.07	*NA*	*NA*	1.0	0.55	0.55	0.07	0.14	0.14
Phone	0.13	0.14	0.14	0.07	0.44	0.07	*NA*	*NA*	0.55	1.0	1.0	0.51	0.14	0.14
Name	0.23	0.07	0.07	0.27	0.08	0.43	*NA*	*NA*	0.07	0.52	0.52	1.0	0.08	0.09
Street	0.28	0.14	0.14	0.28	0.14	0.08	*NA*	*NA*	0.27	0.71	0.27	0.44	1.0	1.0
State	0.07	0.07	0.07	0.14	0.07	0.14	*NA*	*NA*	0.07	0.07	0.07	0.14	1.0	1.0
City	0.07	0.07	0.07	0.28	0.35	0.14	*NA*	*NA*	0.07	0.08	0.08	0.3	1.0	1.0
Style	0.28	0.07	0.07	0.66	0.23	0.66	*NA*	*NA*	0.23	0.23	0.23	0.27	0.07	0.51

Figure 11: Combined confidence-value matrix

elements should match (either directly or indirectly). Since the lightweight domain ontologies are 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 $conf(s,t)$, where $conf^v(s,t)$ either equals $(conf_2(s,t) + conf_3(s,t))/2$ if both the confidence values from data-value characteristics and expected-data values are available or equals $conf_2(s,t)$ if we cannot obtain $conf_3(s,t)$ because of the incompleteness of domain ontologies[16]. Figure 11 shows the matrix that contains the combined confidence values obtained at the end of this step. An NA in the matrix denotes that we do not apply any confidence value between a lexical object set and a nonlexical object set in our approach for schema mapping.

Step 2: *Analyze equivalence classes and their semantic correspondences between S and T.* Based on functional relationship sets, we identify two sets of equivalence classes, E_S in S and E_T in T. We next distinguish the representative equivalence classes in S and T as described in Section 4. Figure 8 shows these contexts after the first phase of context construction for representative equivalence classes of the schemas in Figure 1.

When comparing two representative equivalence classes $e_S \in E_S$ and $e_T \in E_T$ for the second phase of context construction, we take three factors into account: (1) the 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 pair of representative equivalence classes to be compatible, denoted by $(e_T \sim e_S)$, if (1) one confidence value $conf(s', t') \in$

[16]A weight could be assigned with each confidence value from a matching technique by a human expert. The assignment, however, requires expert knowledge on domains as well as the techniques.

$\{conf(s,t) | s \in e_S,\ t \in e_T\}$ is high, and (2) both the importance similarity $sim_{importance}(e_S,\ e_T)$ and the vicinity similarity measure $sim_{vicinity}(e_S,\ e_T)$ are high. The latter two measures together represent the similarity between contexts of e_S and e_T which we obtain after the first phase of context construction as discussed in Section 4. Given an experimentally determined threshold, th_{conf},[17] we calculate $sim_{vicinity}(e_S,\ e_T)$ and $sim_{importance}(e_S,\ e_T)$ based on the following formulas. In the formulas, $Cont_{e_S}$ ($Cont_{e_T}$) denotes the set of object and relationship sets for the context of e_S (e_T) and $Cont^O_{e_S}$ ($Cont^O_{e_T}$) denotes just the set of object sets in $Cont_{e_S}$ ($Cont_{e_T}$).

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

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

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 first-phase context of e. When the number of schema elements is largely different, [MBR01] reports that it is difficult to determine the similarity based only on the singular measure, $sim_{vicinity}$. Thus, we add $sim_{importance}$, which is based on a conceptual-analysis technique discussed in [CAFP98] to help measure the context similarity from an additional perspective.

The comparison between equivalence classes in the target T and the source S provides the "guess" about semantic correspondences. Thus, by using the algorithm in Figure 9, we proceed with the second phase of context construction for representative equivalence classes in S and T. Figure 10 shows the modified contexts for representative equivalence classes in our running example.

At this point, we are finished with the top-level comparison between S and T. We are now 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 ($e_T \sim e_S$), which represents two compatible representative equivalence classes determined in Step 2, we use the combined confidence values between object sets to determine semantic correspondences between

[17]For any application domain, the computed confidence values tend to converge to a specific high measure for element matches between two schemas. Thus we use a cross-application threshold value.

	MLS	*baths*	*beds*	*category*	*SQFT*	*loc._desc.*	*basic_feat.*	*address*
House	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	0.11	*NA*
Bathrooms	0.14	0.92	0.66	0.09	0.27	0.07	*NA*	0.14
Bedrooms	0.14	0.65	0.92	0.09	0.14	0.07	*NA*	0.14
MLS	1.0	0.22	0.13	0.07	0.14	0.23	*NA*	0.3
Square_feet	0.14	0.27	0.14	0.07	1.0	0.07	*NA*	0.14
Water_front	0.07	0.07	0.07	0.14	0.07	1.0	*NA*	0.08
Golf_course	0.07	0.07	0.07	0.14	0.07	1.0	*NA*	0.07
Address	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	0.12	*NA*
Street	0.28	0.14	0.14	0.28	0.14	0.08	*NA*	1.0
State	0.07	0.07	0.07	0.14	0.07	0.14	*NA*	1.0
City	0.07	0.07	0.07	0.28	0.35	0.14	*NA*	1.0
Style	0.28	0.07	0.07	0.66	0.23	0.66	*NA*	0.51

Figure 12: Confidence-value matrix between contexts for $\{MLS\} \sim \{House, MLS\}$

	agent	*fax*	*phone_day*	*phone_evening*	*name*	*location*
Address	0.11	*NA*	*NA*	*NA*	*NA*	*NA*
Agent	0.98	*NA*	*NA*	*NA*	*NA*	*NA*
Fax	*NA*	1.0	0.55	0.55	0.07	0.14
Phone	*NA*	0.55	1.0	1.0	0.51	0.14
Name	*NA*	0.07	0.52	0.52	1.0	0.08
Street	*NA*	0.27	0.71	0.27	0.44	1.0
State	*NA*	0.07	0.07	0.07	0.14	1.0
City	*NA*	0.07	0.08	0.08	0.3	1.0

Figure 13: Confidence-value matrix between contexts for $\{agent\} \sim \{Agent\}$

object sets. Figure 12 and Figure 13 show the confidence-value matrices we base to discover object-set matches between the contexts for ($\{MLS\} \sim \{House, MLS\}$) and ($\{agent\} \sim \{Agent\}$) respectively. We first discover object-set matches between $Cont^O_{e_S}$ and $Cont^O_{e_T}$ that hold with the highest confidence value ($conf = 1.0$). For all remaining unsettled object sets in $Cont^O_{e_S}$ and $Cont^O_{e_T}$, we find a best possible match using an injective-match settling algorithm so long as the confidence of the match is above the threshold, th_{conf}. As an example, after we obtain semantic correspondences between object sets within the contexts of $\{MLS\}$ and $\{House, MLS\}$ based on the perfect confidence values, we can further settle another two object-set matches by applying the injective-matching settling algorithm based on the high confidence values $conf(beds, Bedrooms)$ and $conf(baths, Bathrooms)$, which are above the threshold value $th_{conf} = 0.8$.

For each of the object-set semantic correspondence, we keep the manipulation operations obtained by determining the expected-data-values patterns, which are required to transform source elements into virtual source object sets that directly match with target object sets. For example, we keep the *Decomposition* operations identified by the expected-data-values patterns between *location* in Schema 1 and *Street*, *City*, and *State* in Schema 2. We will use these operations to specify mapping expressions for indirect matches in Steps 3 and 4.

With the available semantic correspondences between object sets in S and T, we further discover matches between relationship sets. We limit the recognition of most relationship-set matches within the contexts of compatible representative equivalence classes between S and T. However, for relationship sets that go between the contexts of compatible representative equivalence classes, we identify semantic correspondences globally without the limitation of contexts. The recognition of a relationship-set match starts by locating a relationship set r_t in T. Then, based on the object sets O_{r_t} connected by r_t, we can locate a set of object sets that correspond to O_{r_t} in S, from which we either locate or derive a relationship set r_s that corresponds to r_t.

More particularly, between the contexts of two compatible representative equivalence classes, e_T in the target T and e_S in the source S, we first recognize semantic correspondences for relationship sets that connect object sets in e_T with relationship sets in or views over the context $Cont_{e_S}$ of e_S. As an example, given $(\{House, MLS\} \sim \{MLS\})$, we start processing the relationship set $House$—MLS in the context of $\{House, MLS\}$ in the target. To obtain its corresponding relationship set in the source, we use a *Skolemization* operator to derive a virtual relationship set $House'$—MLS in the context $\{MLS\}$. We next recognize semantic correspondences for target relationship sets each of which connects at least one object set that is in $Cont^O_{e_T}$ but not in e_T with relationship sets in or views over $Cont_{e_S}$. For example, to match with the target relationship set $House$—*Bedrooms* in the context of $\{House,\ MLS\}$, which connects one object set *Bedrooms* that is not in $\{House,\ MLS\}$, we use the *Join* and *Projection* operators to derive a virtual relationship set $House'$—*beds* over the context of $\{MLS\}$ in the source.

After discovering relationship-set matches within contexts of compatible representative equivalence classes, we discover the semantic correspondences for target relationship sets that contains relationships connecting objects in different contexts of matching representative equivalence classes. In our running example, $House$—*Agent* is such a relationship set in the target connecting object sets in the contexts of $\{House,\ MLS\}$ and $\{Agent\}$ in Figure 10(b). With the available object-set correspondence between $House'$ and $House$ and the correspondence between *agent* and *Agent*, we derive a virtual relationship set $House'$—*agent* in the source that corresponds $House$—*Agent* in the target.

We now give all the derivation of all virtual object and relationship sets obtained in Step 3 for our running example. In the derivation, the assignment arrow ($\Leftarrow$) in each step denotes a virtual element on the left derived by applying the algebra expression on the right.

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

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

These derivations are based on correspondences determined by using the confidence-value matrix in Figure 12 between object sets in the context of $\{MLS\}$ and object sets in the context of $\{House, MLS\}$. The structure-matching technique decides that the values for MLS, *baths*, *beds*, $SQFT$ in the context of $\{MLS\}$ of Figure 10(a) directly correspond to the values for MLS, *Bathrooms*, *Bedrooms*, $Square\_feet$ in the context of $\{House,\ MLS\}$ of Figure 10(b) respectively. Based on the available object-set correspondences, the algorithm derives $House'$—MLS, $House'$, $House'$—$baths$, $House'$—$beds$, $House'$—$SQFT$. In addition to the direct object-set matches, the algorithm determines that the values for *location_description* in the source are generalizations of the lot description implied in the values for $Water\_front$ and $Golf\_course$ in the target. Thus, the algorithm derives $House'$—*location_description* based on this indirect match as well as the newly obtained object-set match between $House'$ and *House*. Moreover, since the values for *Street*, *City*, and *State* in the target are split values for values of *address* in the source, the algorithm also derives *House*—$Address1'$, $Address1'$, and $Address1'$—*address*.

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

$agent$—$Address2' \Leftarrow \varphi_{f_{Address2'}}(agent)$
$Address2' \Leftarrow \pi_{Address2'}(agent$—$Address2')$
$Address2'$—$location \Leftarrow \pi_{Address2',location}(agent$—$Address2' \bowtie agent$—$location)$
$agent$—$'fax \Leftarrow \sigma_{key(agent)}(agent$—$fax)$

Based on the confidence-value matrix in Figure 13, the structure-matching technique decides that the objects for *agent* in the source directly correspond to the objects for *Agent* in the target and that the values for *location* in the source indirectly correspond to the values for *Street*, *City*, and *State*, in the target. With these object-set correspondences, the algorithm derives the virtual elements *agent*—$Address2$, $Address2'$ and $Address2'$—*location*. In addition to the correspondence between *agent* and *Agent*, the algorithm also determines the correspondence between fax and Fax because $conf(fax, Fax) = 1.0$ in Figure 13. However, the relationships in *Agent*—Fax in the target are only a subset of the relationships in *agent*—fax in the source because the functional dependency $Agent \rightarrow Fax$ in the target more tightly constrains the relationship set than does the many-many relationship set in the source. As a default, we transform *agent*—fax with the *Selection* operator $\sigma_{key(agent)}$ which selects as many relationships as possible while maintaining the property that *agent* is a key in the new relationship set and thus ensures that the FD $agent \rightarrow fax$ holds. To allow for an alternate to the default, the system alerts the DBA to the constraint violation and lets the DBA specify a different selection condition, if desired.

3. *Derivation of a virtual relationship set between the contexts of* $\{MLS\}$ *and* $\{agent\}$.

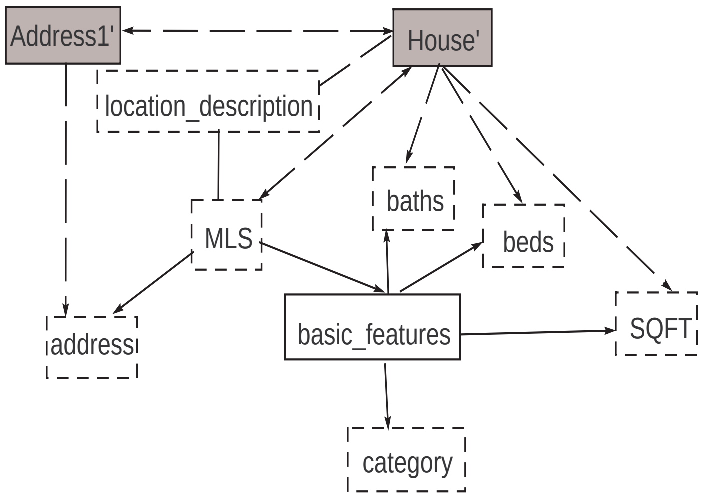
   $House'\text{—}agent \Leftarrow \pi_{House', agent}(House'\text{—}MLS \bowtie MLS\text{—}agent)$

   Note that even though the view derivation happens beyond the limitation of contexts of compatible representative equivalence classes, we still can constrict the search spaces by using available object- and relationship-set matches obtained within the two contexts.

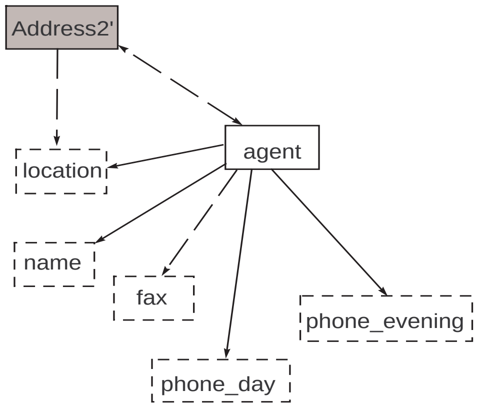
Figure 14 shows the virtual object and relationship sets obtained when detecting relationship-set matches in Step 3. The dashed lines represent virtual relationship sets, and the shaded boxes represent virtual object sets.

Step 4: *Specify mapping expressions for object- and relationship-set matches.* For direct matches, the specification of mapping expressions for mapping elements is straightforward. However, the specification of mapping expressions for indirect matches is nontrivial. 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 for indirect matches discovered in Step 3. Then, at the top level, we derive virtual elements based on schema-level information. We discuss the two levels as follows.

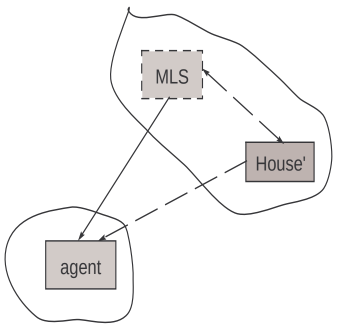
1. *Instance-level derivations.* The derivation of virtual object and relationship sets by applying instance-level information depends on manipulation operations output from the matching technique while searching for expected-data values as described in Section 3. Figure 15 shows the virtual object and relationship sets derived after applying the instance-level information



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

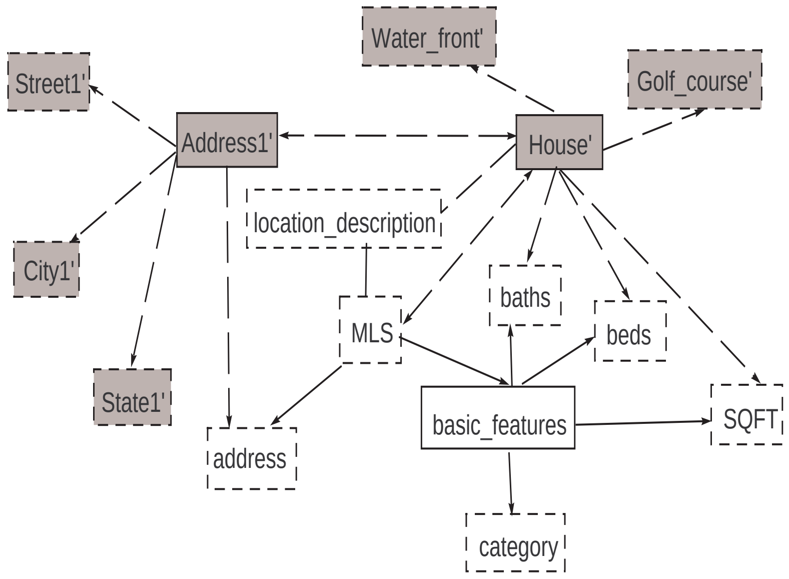


(b) In the Context of $\{Agent\}$

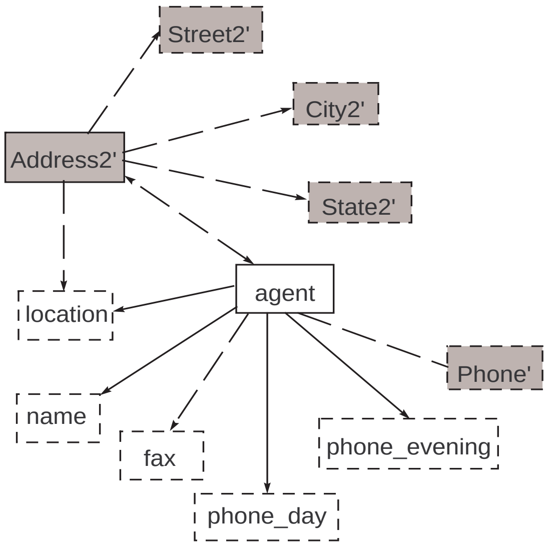


(c) Inter Relationship Sets between the contexts of $\{Agent\}$ and $\{MLS\}$

Figure 14: Discovering object and relationship-set matches



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



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

Figure 15: Derived virtual object and relationship sets based on expected-data values

for our running example. Here, again, shaded boxes represent virtual object sets, and dashed lines denote virtual relationship sets.

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

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

  The *DeBoolean* operators make new virtual relationship sets such that the values in the formed virtual relationship sets use Boolean indicators “Yes”/“No” as values. The three *Decomposition* operators use routines $R^{address}_{Street1'}$, $R^{address}_{City1'}$, and $R^{address}_{State1'}$ to decompose the string values for *address* as values for the new virtual object sets $Street1'$, $City1'$, and $State1'$.

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

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

  The three *Decomposition* operators make virtual relationship sets based on routines $R^{location}_{Street2'}$, $R^{location}_{City2'}$, and $R^{location}_{State2'}$, which decompose the string values for *location* as values for the new virtual object sets $Street2'$, $City2'$, and $State2'$. Indeed, the three routines used here are the same as those used to extract values for $Street1'$, $City1'$, and $State1'$ in the context of $\{MLS\}$. The agent's $Phone'$ values are a union of the *phone_day* and *phone_evening* values.

- *Derivation of virtual object and relationship sets between the contexts of* $\{MLS\}$ *and* $\{agent\}$. The indirect match between *House′—agent* in the source and *House—Agent* in the target does not depend on any manipulation operation derived by applying expected-data values. Thus we do not need to to derive virtual elements between the contexts of $\{MLS\}$ and $\{agent\}$.

2. *Schema-level derivations.* 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 that occur in different context pairs. For example, both objects in $Address1'$ in the context of $\{MLS\}$ and objects in $Address2'$ in the context of $\{agent\}$ in the source correspond to 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 one-to-one relationship sets between source elements in V_S and target elements in Σ_T.

   - *Derivation of virtual object sets for indirect object-set matches.*

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

     It is an object-identity problem to merge any objects in $Address1'$ and $Address2'$ as objects in $Address'$. Recognizing object identity is beyond the scope of this paper, we thus assume that we have a resolution or that duplicates do not matter. If the *Selection* operator is needed (*Selection* is not needed when Schema 2 is the target, as we currently are assuming), 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.

   - *Derivation of virtual relationship sets for indirect relationship-set matches.*

     $Address'\text{—}Street' \Leftarrow \rho_{Address1' \leftarrow Address', Street1' \leftarrow Street'} Address1'\text{—}Street1$
     $\qquad \cup\ \rho_{Address2' \leftarrow Address', Street2' \leftarrow Street'} Address2'\text{—}Street2'$
     $Address'\text{—}City' \Leftarrow \rho_{Address1' \leftarrow Address', City1' \leftarrow City'} Address1'\text{—}City1$
     $\qquad \cup\ \rho_{Address2' \leftarrow Address', City2' \leftarrow City'} Address2'\text{—}City2'$
     $Address'\text{—}State' \Leftarrow \rho_{Address1' \leftarrow Address', State1' \leftarrow State'} Address1'\text{—}State1$
     $\qquad \cup\ \rho_{Address2' \leftarrow Address', State2' \leftarrow State'} Address2'\text{—}State2'$
     $agent\text{—}'Phone' \Leftarrow \sigma_{key(agent)}(agent\text{—}Phone')$

     The last derivation forces the participation constraint of *agent* in the relationship set

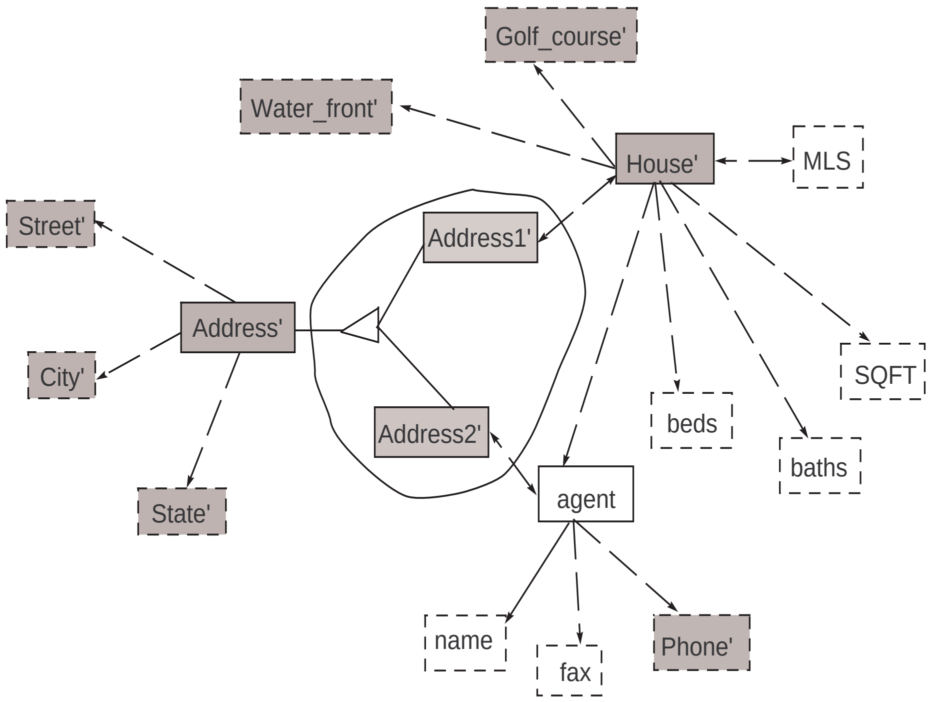


Figure 16: Source elements in the source-to-target mapping between Schema 1 (source) and Schema 2 (target)

> $agent$—$'Phone'$ to match with the functional constraint of $Agent$ in the relationship set $Agent$—$Phone$.

Figure 16 shows the source elements, which are object- and relationship-sets outside of the enclosed area, in the source-to-target mapping between S and T of our running example. The open white triangle denotes generalization/specialization. We use this notation to illustrate that the objects in $Address'$ are union of the objects in $Address1'$ and $Address2'$. Note that the object set $Address$ in the target directly matches the virtual object set $Address'$ in the source. The relationship sets $House$—$Address$ and $Agent$—$Address$ in the target, however, match with $House'$—$Address1'$ and $agent$—$Address2'$ respectively. Thus both the object set $Address'$ and the relationship sets $House'$—$Address1'$ and $agent$—$Address2'$ are in the source-to-target mapping but neither $Address1'$ nor $Address2'$ is in the mapping.

Step 5: *Output a source-to-target mapping.* All the derivation in Step 3 and 4 generates the virtual object and relationship sets for the source-to-target mapping for our running example. At this point, there is a one-to-one mapping between source and target object and relationship sets. Thus, for example, $Address'$ maps $Address$, $House'$—$Address1'$ maps to $House$—$Address$, and $agent$—$'fax$ maps to $Agent$—Fax.

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.[18] 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 Examples

We applied the mapping algorithm explained in Section 5 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.

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 eight direct matches correctly, but it also misclassified the source object set *address*

[18]We manually constructed the input for all applications in the representation required by the algorithm.

Run Nr.	Number of Matches (N)	Number Correct (C)	Number Incorrect (I)	Recall %	Precision %	F-Measure %
1 (WS)	30	18	3	60%	86%	71%
2 (WCS)	30	18	1	60%	95%	73%
3 (WES)	30	30	0	100%	100%	100%
4 (WCES)	30	30	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

(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*. Also, the algorithm picked up a direct match between *phone_day* and *Phone* but lost the correspondence between *phone_evening* and *Phone*. In the first run, the algorithm successfully discovered 10 of the 22 indirect matches. For example, by using the *Skolemization* operator, the algorithm matches the object identifiers for a virtual nonlexical *House′* based on the values in *MLS* with object identifiers in *House*. The mapping 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 the *Skolemization* operator two times to compute virtual objects in *Address1′* and *Address2′* that match 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 two incorrect matches between *address* and *Style* and between *house′—address* and *House—Style* discovered based on terminological relationships 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 (N)	Number Correct (C)	Number Incorrect (I)	Recall %	Precision %	F-Measure %
1 (WS)	25	15	3	60%	83%	70%
2 (WCS)	25	15	1	60%	94%	73%
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 in Table 2. In the first run for the second test, the algorithm correctly discovered eight direct and seven of 17 indirect matches, but it also misclassified *Style* and *House—Style* by matching them with the target object set *address* and a virtual relationship set *house′—address*. Because our approach used an injective-matching settling algorithm to obtain direct matches, the algorithm matched *Phone* in the source with *phone_day* in the target but did not discover that the phones in *phone_evening* are also a subset of values in *Phone*. In the second run, by adding the analysis of data-value characteristics, the incorrect matches between *Style* and *address* and between *MLS—Style*, which is a virtual relationship set, and *MLS—address* output based on terminological relationships 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 such as the semantic correspondence between *address* in the target and *Street*, *City*, and *State* in the target and appropriately produced operations consisting of a combination of *Composition*, *Join*, *Projection*, and *Selection*. The *Selection* operator sorted out the addresses composed from *Street*, *City*, and *State* based on the two relationship sets *House—Address* and *Agent—Location* in Schema 2. Moreover, 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, however, the *Selection* operator fails to sort out the *Phone* values.[19]

[19]Even humans could not sort out this anomaly without the help of keywords.

6.2 Real-World Examples

We considered three real-world application domains: *Course Schedule*, *Faculty*, and *Real Estate* to evaluate our approach. We used a data set downloaded from the homepage of a schema-matching approach [DDH01], Learning Source Descriptions (LSD), for these three domains, 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; the source schemas for *Real Estate*, however, are not.

Domain	Number of Sources	Number of ObjSets	Number of RelSets	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

For testing these real-world domains, we decided to let any one of the schema graphs for a domain be the target and let any other schema graph for the same domain be the source. We decided not to test any single schema as both a target and a source. Since for each domain there were five schemas, we tested each domain 20 times. Altogether we tested 60 target-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. Altogether we processed 240 runs. Table 4 shows a summary of the results for the real-world data using all four techniques together.

In the *Faculty* domain, the five schemas happen to be exactly the same—same structure with the same names. The values, however, are different since the faculties are at different universities. The matching algorithm correctly identified all matches. For all four runs on the *Faculty* domain every measure (recall, precision, F-measure) was 100%. Since the five source schemas are the same, but the data instances collected for each object set are vastly different, we assigned a higher weight for w_S than w_V so that schema-level information would dominate. Thus, we should expect a 100%

match.[20]

In the *Course Schedule* domain, there were indirect relationship-set matches that required manipulations using *Join*, *Skolemization*, and *Projection* operators. For the *Course Schedule* domain, 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 four techniques, the correctly recognized mapping elements included 382 of 407 direct and 72 of 83 indirect matches. The incorrectly classified six mapping elements included four direct and two indirect matches. For direct matches, the precision, recall, and F-measures achieved 99%, 94%, and 96%; for indirect matches, the precision, recall, and F-measure achieved 98%, 88%, and 92%. Even when values for lexical object sets were not available in the first run, since most of the indirect matches appeared in this domain are largely dependent on schema-level information, the mapping algorithm correctly identified 376 direct and 76 indirect matches for this domain.

Application	Number of Matches (N)	Number Correct (C)	Number Incorrect (I)	Recall %	Precision %	F-Measure %
Course Schedule	490	454	6	93%	99%	96%
Faculty	540	540	0	100%	100%	100%
Real Estate	876	820	92	94%	90%	92%
All Applications	1906	1814	98	95%	95%	95%

Table 4: Results for real-world examples

The *Real Estate* domain exhibited several indirect object- and relationship-set matches. There are four cases of *Merged/Split Values*, 48 cases of *Subsets/Supersets*, and 10 cases of *Object-Set Name as Value.* 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 performance reached 73% recall, 67% precision, and an F-measure of 70%. In the second run, the use of data-value characteristics improved the performance, but the measures were still below 80%. By applying expected-data values in the last two runs, however, the performance improved dramatically. The

[20]Since we did not choose these schemas, we feel justified in following our matching procedure and obtaining the 100% match. For curiosity's sake, however, we decided, contrary to our procedure, to lower the weight of w_S from our standard of 0.8 when the schema-level information is to be emphasized to our standard of 0.5 when neither schema-level information nor instance-level information is to be emphasized and to our standard of 0.2 when instance-level information is to be emphasized. In both cases, we still obtained 100% for our F-measure. This is because our expected-values technique remains strong enough to correctly match the schemas. However, when we also removed our expected-values technique and kept our value-characteristic technique to "inappropriately" give it emphasis (i.e. when we applied WCS as explained for Table 2), we obtained an F-measure of only 74% for $w_S = 0.5$ and an F-measure of only 66% for $w_S = 0.2$. To further attempt to de-emphasize name matching, we also let $w_S = 0.0$, which entirely removes name matching except in the case of non-lexical object sets, and obtained an F-measure of 50%.

F-measures reached 91% in the third run and reached 92% by using all four techniques as Table 4 shows. The correctly classified 820 mapping elements included 417 of 453 direct matches and 403 of 423 indirect matches. The incorrectly classified 92 mapping elements included 24 direct and 68 indirect matches. Thus, for the direct matches, the precision, recall, and F-measure achieved 95%, 92%, and 93%; and for the indirect matches, the precision, recall, and F-measure achieved 86%, 95%, and 91%.

Our process successfully found all the indirect matches for *Merged/Split Values* and *Object-Set Name as Value*. For *Subsets/Supersets*, our process correctly found all the indirect matches related to 44 of 48 cases of *Subsets/Supersets* and incorrectly declared four extra *Subsets/Supersets* cases. 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 clearly incorrect cases happened when the algorithm unioned (selected) office and cell phones 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 virtual elements derived for *Path as Relationship Sets*. Moreover, the results show that by adding our technique of using expected-data values, the performances are dramatically better even for domains, 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.[21]

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.[22] Even though

[21]We, however, keep the technique in our approach because we believe that it is able to make contributions to schema mapping based on the results of SEMINT [LC00]. Even though it did not help very much in the three application domains presented in this paper, it did work well in another application domain when we tested the approach in our early work [EJX01].

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

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* domain. Thus, although our raw performance numbers are an improvement over LSD [DDH01], we do not try to draw any final conclusion.

One possible limitation to our approach is the need to construct a domain ontology for an application domain. Currently, we manually construct these domain ontologies. As we explained in Section 3, however, these domain ontologies are lightweight, are relatively easy to construct, and need not be complete. Furthermore, because of the multiple techniques applied and various kinds of information exploited, domain ontologies are not necessary as input for our approach. As the experiments show, our approach achieved good performance for the domains *Course Schedule* and *Faculty* without exploiting domain ontologies. The *Real Estate* domain, however, improved about 20% in its F-measure by adding a domain ontology as input. [23] The dramatic increase for the *Real Estate* domain appeared to have happened mainly because of two reasons: (1) many jargon terms that name schema elements in the *Real Estate* domain are not rewritable in terms understood by WordNet, and (2) the usage of value characteristics was unable to exploit statistical features of values in the domain. Hence, rather than being totally dependent on domain ontologies, our approach is flexible and is able to trade off the performance of various techniques to produce source-to-target mappings.

Regarding the creation of domain ontologies for new domains, many values, such as dates, times, and currency amounts are common across many application domains and can easily be shared. Furthermore, it is possible to make use of learning techniques to collect a set of informative and representative keywords for domain concepts in domain ontologies. Thus, without human interaction except for some labeling, we can make use of many keywords taken from the data of the domain itself and thus specify regular-expression recognizers for the domain concepts at least in a semi-automatic way. Since domain ontologies appear to play an important role in indirect

[23]The real-estate domain ontology is composed of about 50 concepts and 25 relationship sets. Most concepts have about five regular expression rules with each of them. There are about 10 concepts that are relatively more complex, each of which has about 10 regular expressions.

matching, finding ways to semi-automatically generate them is a goal worthy of some additional work.

The another problem of our current implementation is the use of thresholds. (Thresholds often cause problems in experimental work.) The parameters and thresholds may work well across applications, but we only know for certain that they work in the applications we present and analyze. Tuning performance parameters, however, requires both expert knowledge of the techniques and application domains. As [MBR01] points out, auto-tuning of parameters in schema mapping is an open problem. In the future, we plan to add features to help users tune parameters in our mapping approach.

7 Related Work

[RB01] provides a survey of several schema mapping systems. We do not repeat this work here, but instead describe work related to our approach from two perspectives: (1) work on discovering direct matches for schema elements, and (2) work on discovering indirect matches for schema elements.

Direct Matches. Most of the approaches [BCV99, BM01, BM02, CAdV01, DDH01, DMD+03, DR02, EJX01, HC03, KN03, LC00, MBR01, MGMR02, MZ98, PTU00] to automating schema mapping focus only on generating direct matches for schema elements.

- In some of our previous work [EJX01], we experimented using schema-level and instance-level information to help identify direct matches. In this paper, we extend this work to generate source-to-target mappings that contains both direct and indirect object- and relationship-set matches.

- As in our approach, the LSD system [DDH01] and its extension GLUE [DMD+03] apply a meta-learning strategy to compose several base matchers, which consider either data instances, or schema information. LSD and GLUE largely exploit machine learning techniques. There are two phases in each system: training and testing. In the training phase, the two systems require training data for each matchable element in a mediated schema 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 cross-application 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 in LSD and GLUE. (3) To combine techniques, we let structure features guide the matching based on the results from multiple kinds of independent matches. When porting to new application domains, we manually predefine domain ontologies, which can work for computations of source-to-target mappings between source and target schemas within domains. Especially, we can incrementally polish the domain ontologies for newly available schemas to improve mapping performances. LSD and GLUE could improve their performance either by marking a large amount of domain-dependent training data or by supervising the training in a clever way using active-learning techniques. However, usually the sample data for each matching element of the mediated schema in multiple learners is not easy to collect and is tedious to label, and how to guide the learning in active learning for different learning models is not trivial either. Active learning may need many labeled examples to "bootstrap" learners so that it makes good estimates about which unlabeled examples are useful, and it also takes much effort to choose initial training examples.

- COMA [DR02] is used as a framework to evaluate the effectiveness of different individual matchers and their combinations. It also provides a matcher aiming at reusing results from previous match operations. The results obtained by COMA show that combining matchers is superior to using any individual matcher. This supports the composite methodology applied in our approach.

- 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. It is flexible to port to new application domains because of the application of learning-based techniques as in LSD. However, SEMINT clusters attributes in one data source first, and then trains a classifier to classify each attribute in another source into a cluster of attributes, not a single attribute, in the first source. The tool reduces the search space for mapping elements although the result search space is still huge and requires human interaction to resolve the correct matches.

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

- ARTEMIS [CAdV01], DIKE [PTU00], and Cupid [MBR01] exploit auxiliary information such as synonym dictionaries, thesauri, and glossaries. All their auxiliary information is schema-level—it does not consider data instances. In our approach, the auxiliary information, including data instances and domain ontologies, provides 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.

- [BM02] describes a system, called Automatch, that uses primarily Bayesian learning to acquire probabilistic knowledge from training examples and stores the knowledge in an attribute dictionary. The acquired knowledge is later exploited to compute mappings between two schemas. Our mapping algorithm also applies learning techniques to acquire knowledge and stores the knowledge in a knowledge base. The knowledge base as well as domain ontologies later are exploited to compute source-to-target mappings. Our approach, however, is able to compute indirect matches that are not considered in [BM02].

- [MGMR02], [KN03], and [HC03] apply statistical analysis techniques as well as graph matching algorithms. None of the three approaches supports capturing indirect semantic correspondences among attributes. However, the three techniques are complementary to the techniques in our approach, and they could be applied as individual matching techniques in our extensible framework.

Indirect Matches. Some work on indirect matches is starting to appear [BE03, DLD$^+$04, DMD$^+$03, 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 nonlexical object set and discard *Street*, *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*, *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 iMAP system described in [DLD+04] is similar to our composite approach. To the best of our knowledge, it is the only other work like ours to automate computations of both direct and indirect matches. iMAP and our work were developed independently. The techniques applied and the knowledge exploited in their approach are complementary to those in our approach.

- The Clio project [MHH00, MHH+01] is a system for managing and facilitating the complex tasks of heterogeneous data transformation and integration. The objective of Clio is to support the generation and management of schemas, correspondences between schemas, and mappings between schemas, which are specified as queries similar to the way we specify mappings. Clio was one of the earliest projects to both recognize the need and provide support for indirect mappings by allowing a user to specify that target values can be created from a set of related values in a source. An extensive tool set aids users semi-automatically generate mappings using an interactive mapping creation paradigm based on value correspondences. A DBA, however, is responsible for inputting most of the value correspondences. Clio and our mapping techniques are independently implemented and are complementary. Our mapping techniques could help Clio discover both direct and indirect matches semi-automatically. On the other hand, our approach could take advantage of the GUI provided by Clio so that a DBA can easily be involved in schema mapping.

- Although the main focus of GLUE [DMD+03] is on machine learning techniques that result in direct mappings, [DMD+03] does include a section on CGLUE, Complex-GLUE, which addresses indirect mappings. The indirect mappings, however, are a small subset of the potential indirect mappings, covering only union-constructed 1:n mappings in an ISA hierarchy in which the children of any taxonomic node are assumed to be mutually exclusinve and exhaustive. Although the subset of the potential mappings is small, the case they consider is quite common. The experimental work in [DMD$^+$03] shows that the accuracy of their approach is reasonably good and has promise.

- [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 complementary to the work in [BE03].

8 Conclusions and Future Work

We presented a composite approach for automatically discovering both direct matches and many indirect matches between sets of source and target schema elements. In our approach, 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* and *De-Boolean* conversions for *Object-Set Names as Value*. 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 domain-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 about 90% in both recall and precision.

9 Acknowledgements

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

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.

[BM01] J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS 2001)*, pages 108–122, Trento, Italy, 2001.

[BM02] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAISE 2002)*, pages 452–466, Toronto, Canada, 2002.

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

[CAdV01] S. Castano, V. De Antonellis, and S. De Capitani di Vimercati. Global viewing of heterogeneous data sources. *Transactions on Data and Knowledge Engineering*, 13(2):277–297, 2001.

[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.

[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 (SIGMOD 2001)*, pages 509–520, Santa Barbara, California, May 2001.

[DLD$^+$04] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex matches between database schemas. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*, Paris, France, June 2004.

[DMD$^+$03] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the semantic web. *VLDB Journal*, 12:303–319, 2003.

[DR02] H. Do and E. Rahm. COMA—a system for flexible combination of schema matching approaches. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB2002)*, pages 610–621, Hong Kong, China, August 2002.

[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 L. 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.

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

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

[HC03] B. He and K. Chang. Statistical schema matching across web query interfaces. In *Proceedings of SIGMOD 2003*, 2003.

[Hew00] K.A. Hewett. An integrated ontology development environment for data extraction. Master's thesis, Brigham Young University, Provo, Utah, April 2000.

[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.

[KN03] J. Kang and J.F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD 2003)*, pages 205–216, San Diego, California, June 2003.

[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.

[LEW00] S.W. Liddle, D.W. Embley, and S.N. Woodfield. An active, object-oriented, model-equivalent programming language. In M.P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object-Oriented Data Modeling*, pages 333–361. MIT Press, Cambridge, Massachusetts, 2000.

[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.

[MGMR02] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and it application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002)*, pages 117–128, San Jose, California, 2002.

[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 *FUSSION 99*, 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, 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*, 10:334–350, 2001.