Integrating Structured Data Using Property Precedence

By

©Safwan Mustafa

A Thesis submitted to the

School of Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Science

Computer Science Department

Memorial University of Newfoundland

Aug 2015

St. John's Newfoundland and Labrador

ABSTRACT

Data integration systems offer uniform access to a set of autonomous and heterogeneous data sources. One of the main challenges in data integration is reconciling semantic differences among data sources. Approaches that been used to solve this problem can be categorized as schema-based and attribute-based. Schema-based approaches use schema information to identify the semantic similarity in data; furthermore, they focus on reconciling types before reconciling attributes. In contrast, attribute-based approaches use statistical and structural information of attributes to identify the semantic similarity of data in different sources. This research examines an approach to semantic reconciliation based on integrating properties expressed at different levels of abstraction or granularity using the concept of property precedence. Property precedence reconciles the meaning of attributes by identifying similarities between attributes based on what these attributes represent in the real world. In order to use property precedence for semantic integration, we need to identify the precedence of

attributes within and across data sources. The goal of this research is to develop and evaluate a method and algorithms that will identify precedence relations among attributes and build property precedence graph (PPG) that can be used to support integration.

ACKNOWLEDGEMENTS

This thesis could not be possible without the help of my supervisor Dr. Jeffrey Parsons; he provided me great insights and inspirations without which this thesis is impossible to come to life. I really appreciate your help and support for my study and research. I also would like to thank all computer science staff and faculty member who offered me assistance in my graduate study.

Table of Contents

List of Figures

List of Tables:

1 Introduction

Data is scattered everywhere. It can be isolated in local databases or shared on the World Wide Web. The invention of the Internet and emergence of the World Wide Web makes it easier for people and organizations to access and share data all around the world. The shared data can be structured, semi-structured or unstructured. However, the amount of structured data in the web increased after the World Wide Web Consortium (W3C) adapted a new vision for the web. This vision is represented in the sematic web(Berners-Lee, Hendler, & Lassila, 2001), and it recommends sharing data in the Resource Description Framework (RDF) data model, which tends to be structure format.

Large enterprises may have dozens or hundreds of disparate and autonomous data sources and these data sources can be found online, on the Web or inside the local network of the enterprise. These sources may vary on multiple dimensions, such as the data model and the query language that they support. As a result of such situations, automating a lot of straight-forward processes inside

the enterprise or providing business support decisions through Business Intelligence (BI) tools is impossible because of the lack of communication between these data sources. To provide such communication and integration between data sources, data integration can be used. Moreover, modern enterprises are more interested in establishing such communication with other data sources outside the boundaries of their systems to provide complete services to customers and to enhance the capabilities of the (BI) tools by providing more useful information from multiple sources.

The traditional way to solve the data integration problem was to use data warehousing, which is based on transforming data from multiple sources and store it in a central database through three main processes: extracting, transforming and loading (ETL). Data warehouse concept was originally developed to perform deeper analysis for enterprises (Doan, Halevy, & Ives, 2012). But industry adapted data warehousing as an approach for data integration. This approach for data integration did not last long as a standard due to a lot of disadvantages such as high cost, long implementation time and out of date data.

In the late 1990's, a virtual integration approach or Enterprise Information Integration (EII) - as it is known in the industrial field - moved from labs to the industry area (Halevy, Rajaraman, & Ordille, 2006). This approach does not require transforming data and storing them in a central database, but rather provides a uniform interface called mediated schema or global schema for posing queries. This mediated schema shows data as if it is stored in a single database.

Nowadays, the concept of data integration refers to virtual integration approach for data integration. Halevy (2001) defined data integration systems as systems that provide a uniform query interface to a multitude of autonomous and heterogeneous data sources.

1.1 Motivation

To build a data integration system, the application designer needs to specify the mediated schema and sources description. A mediated schema captures relevant domain aspects and data source descriptions that will be used to link sources with the mediated

schema. Each source description contains a schema that describe the source content and a semantic mapping that maps relevant attributes in the data source to the corresponding attributes in the mediated schema. This mapping process requires data semantic reconciliation. Parsons and Wand (2003) argue that the proposed approaches for attribute-based semantic reconciliation do not use the real semantic information that can be extracted from attributes or the data itself, rather use statistical or structural information about the attributes. Therefore, they propose a new approach for semantic reconciliation based on the concept of property precedence.

Creating a semantic mapping manually is a labor-intensive and error-prone process. Automating this process is one of the main challenges in data integration field.

To construct matches between attributes using the real semantics of data and to automate the creation of sematic matches between attributes my thesis will focus on the following: Examine property precedence approach for semantic reconciliation, Use a bottom up approach (subsumed scope of attributes and data) for identifying

the precedence relations between attributes, Develop and implement algorithms for building property precedency graph (PPG) that can be used as basis for the integration process, Show how to reformulate the queries posed against DBpedia data source using PPG.

Building PPG automatically to enable the use of property precedence approach in semantic reconciliation is the main contribution for my research.

1.2 Thesis outline

In the next chapter I provide a comprehensive review of the literature in information integration. Subsequently, I discuss the problem of semantic heterogeneity reconciliation and introduce Property Precedence approach for semantic reconciliation. In Chapter 4 I show how to identify all local precedence relations between attributes in data source and how to use the precedence relations in reformulating queries. In Chapter 5 I discuss the results of implementing the algorithms used to build property precedence graph on DBpedia data source.

2 Chapter 2: Literature Review

2.1 Data Integration

Data integration systems are widely used in applications that need to query multiple independent and heterogeneous data sources. According to Ventana research value index (San ,2012) more than half of organizations integrate six or more data sources. Moreover, they projected that this number should reach to 68% of organizations by 2013. The values index results are based on a survey that target 13 vendor with relevant data integration products.

The main purpose of data integration systems is to fetch data on demand from disparate data sources and get the most up-to-date version of data. Solving the data integration problem is hard for several reasons: data sources may run on different hardware, data sources may use different data models, each data source may support a different query language, data sources may have heterogeneous schema and the values of attributes in the different data sources may represented in different ways.

As mentioned in Chapter 1 data warehousing and virtual integration (EII) can be used to solve data integration problem. The main focus on this thesis is on virtual integration. Virtual integration provides users with a global schema (mediated schema) to pose their queries and uses mappings to reformulate the queries to queries over the data sources. Figure 1-1 shows the architecture of virtual integration system.

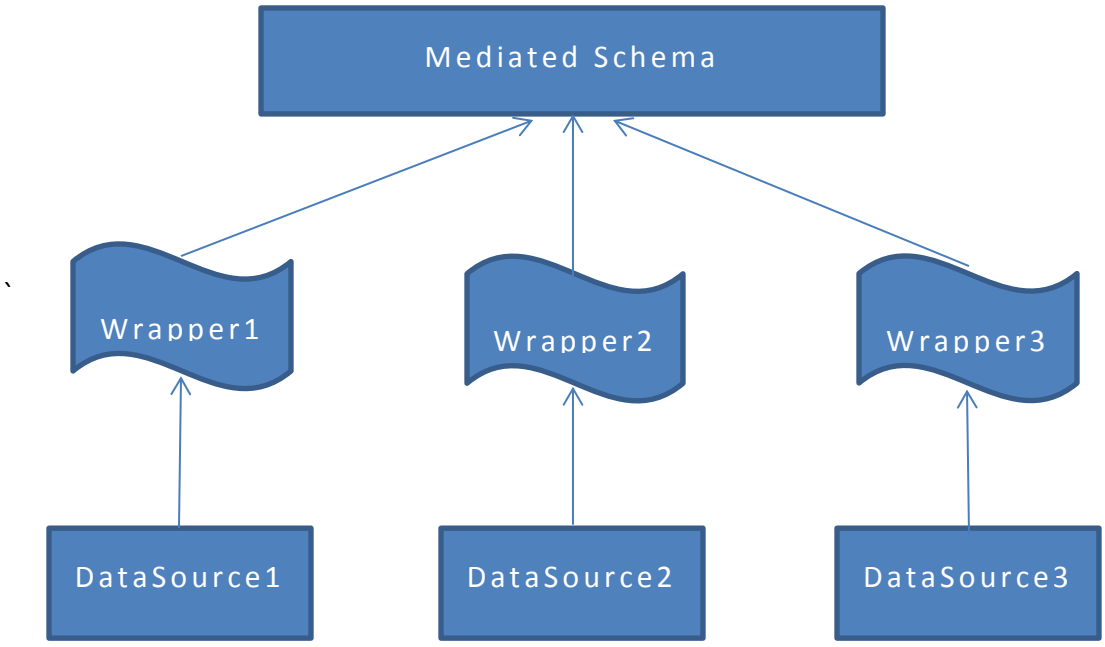

Figure 1-1: Virtual Integration Architecture

On the bottom of figure 1-1, we have the data sources involved in the integration process. Data sources may vary in different dimensions, for example, the data model that they use and the query language that they support. Above each data source we have a program called a wrapper. The wrapper's role is to send queries to data sources, receive answers, and possibly apply some transformation processes on the queries and the received answer. For example, a wrapper for a website would receive queries posed on the mediated schema and transform them to HTTP request to that website, when the answer comes back as a HTML page, the wrapper will extract the query answer tuples from the webpage and send them to the mediated schema.

Users interact with a data integration system by posing their queries over the mediated schema (Global Schema), which provides a generic interface for users to pose queries. A mediated schema is a logical schema that does not store any data. It contains a subset of data sources' attributes that are relevant to the integration application domain. As we explain in the next section, source

description plays the main role in transforming queries from the mediated schema into queries over the data sources.

2.2 Source Descriptions

In data integration systems users pose their queries over the mediated schema. Query processing in data integration system begins by reformulating each query over the mediated schema into a set of queries over the data sources. Then these queries are executed efficiently with an engine that creates execution plans for queries over data sources. To accomplish the reformulation process, data integration system must know which sources are available, which data exists in each source and how to access these sources. A source description in the data integration system provides such information.

Source descriptions also contain other information, such as information to optimize queries for data sources, what type of query the source supports and the access pattern limitation that will prevent illegal access patterns. Such information will help in transforming the logical query plan into an executable plan.

2.3 Schema Mapping

Schema mapping is one of the main components in source descriptions (Halevy, Rajaraman, & Ordille, 2006) . It explains what data exists in the data source and how the terms in this source schema are related to the terms in another target schema, in other words, it is a set of high level expressions that describe the relationship between two schemas regarding the implementation (mediated schema and source schema in data integration system).

Schema mapping is used in reformulating queries over mediated schema into a set of appropriate queries over data sources. The result of the reformulation process is called a logical query plan. In addition to using schema mapping to support the query reformulation process in data integration systems, schema mapping is used in data exchange and data warehouse. In data warehouse schema mapping is used to map data from the source schema to the target schema, which is usually a data warehouse. Furthermore, it may be used in merging two autonomous data sources since it provides the semantic relationship between these two sources.

As a result of the query reformulation process we get the logical query plan, which is a query expression that is related only to the relations on the data sources. The logical query plan will be later optimized for efficiency. The problem of creating a logical query plan is related with answering query over views (Levy, Rajaraman, & Ordille, 1996).

Schema mapping should be able to handle different types of heterogeneity and reconcile these heterogeneities by discovering the semantics between related elements. Even if these schemas refer to the same domain, such heterogeneity exists because data sources were not created for the exact purpose and their schema is built by different designers. Therefore each schema has a different view and naming scheme for schema elements. Heterogeneity may be in schema level or in data level.

Schema level heterogeneity may takes different forms, such as:

- Different names for relations and attributes to refer to the same real world object and its properties. For example, using

the attribute “rating” or the attribute “classification” and “sex” or “gender”.

- Multiple attributes in one schema correspond to a single attribute in another. For example, “first_name” attribute and “last_name” attribute in one schema refer to the “name” attribute in another.
- Tabular organization is when the number of relations is different in each schema, the coverage and details of each data source are also different, since each data source created for different purpose (Doan, Halevy, & Ives, 2012).
- Entities or attributes may have the same name but this does not mean that they refer to the same concept.

Data level heterogeneity may take different forms, such as:

- Different measurements units. For example, GPA using letter scale versus numerical values.
- Different strings used to refer to the same real world concept, like using HP as company name or Hewlett Packard.

Reconciling heterogeneity is a hard process because schemas do not capture all semantics for data. Even if some schemas provide a written description for its tables and attributes, it is still hard to understand the intended meaning of the data. Moreover, schema clues are unreliable since schemas may use the same words to refer to different real-world concepts and semantics can be subjective. Using the priori approach for reconciling heterogeneity, in which database designers follow domain standards for modeling data (e.g., use HL7 for modeling data in health care domain), this is not sufficient solution for reconciling heterogeneity.

Creating standards is not a practical approach since it is hard to specify the boundary of each domain and it is hard to agree on the standards because some organizations already establish their own schemas and follow their own standards. However, such an approach works if there are a small number of attributes and tables, for example, exchanging money between banks or exchange patient information between different health care systems.

2.3.1 Schema Mapping Modeling Languages

The language of schema mapping should be flexible by allowing the addition of new sources easily, and enabling the expression of a wide variety of relations between data sources. It should also be efficient in reformulating queries over the mediated schema. Three different modeling languages for schema mapping: Local-As-View (LAV), Global-As-View (GAV) and Global-Local-As-View (GLAV) which combines features from both LAV and GAV.

Global-As-View (GAV) schema mapping modeling language was used before the LAV approach in data integration systems. It is first introduced in research project TSIMMIS(Garcia-Molina et al., 1997). In GAV each relation (table, class or entity) in the mediated schema (Global schema) is specified as a view over the data sources relations. To reformulate a query using GAV schema mapping, we simply need to unfold the query posed over the mediated schema by replacing the expressions in that query with the corresponding view according to the schema mapping definition.

This modeling language of schema mapping is suitable for systems that are based on a stable number of data sources. Adding a new data source for such systems will affect the definition of the mediated schema mapping, which is defined as a view over the data sources. As a result, adding such sources will require redefining the affected definitions. MOMIS (Mediator envirOnment for Multiple Information Sources) (Beneventano et al., 2000) system and The TSIMMIS (Standford-IBM Manager of Multiple Information Sources) (Garcia-Molina et al., 1997) are examples of data integration systems that use GAV schema mapping modeling language.

Local-As-View(LAV) schema mapping modeling language was one of the main contributions of the information manifold project for data integration (Kirk, Levy, Sagiv, & Srivastava, 1995) which includes integrating more than 100 data sources, most of them on the World Wide Web. In LAV, each relation in the source schema is described as a view over the mediated schema. This modeling language for schema mapping is suitable for systems that have a stable mediated schema. Adding data sources using LAV does not require modifying schema mapping relations on other sources and it does not require modifying the mediated schema, since schema mapping for each

source is defined independently from other data sources. However, query reformulation using LAV is more complicated than GAV so this leads to increase the computing complexity for reformulation process. The IM (Information Manifold) data integration system(Manolescu, Florescu, & Kossmann, 2001) is an example of systems that use LAV.

Global-Local-As-View (GLAV) was first introduced in Navigational Plans For Data Integration (Friedman, Levy, & Millstein, 1999). GLAV combines the two approaches mentioned above since it defines mapping by describing the relation between views over data sources and views over the mediated schema. This language has mainly been proposed to model data sources that are websites which need the expressive power of both GAV and LAV. GLAV is more expressive than both previous schema mapping languages and the query rewriting is not harder than LAV (Friedman et al., 1999).

2.3.2 Schema Mapping Generation

To build schema mapping we begin by specifying semantic matches between schemas and then elaborate them into mapping. Dividing

the process into two steps, schema matching and schema mapping, will reduce the overall complexity of the whole process. We begin by schema matching because it is easier to obtain matches from designers based on their domain knowledge; such matches require designers to reason about individual schema elements. Matches can be found by designers in a visual interface that represents both schemas (source and mediated) elements. However, such a technique is tedious, error-prone, labor-intensive and time consuming in case of a large number of schemas with a large number of tables and attributes.

Automatic or semi-automatic schema matching system provides a more convenient way for schema matching, it provides designers with a set of matches and gives them the ability to refine, confirm or reject matches. In contrast, manual schema matching systems require designers to build such matches from the scratch. Automatic matching can be done using different heuristics but it does not guarantee accurate matches. One type of heuristic might be based on the similarities between the schema element's names. Another type of heuristic might be based on the similarities between schema element's values or on how attributes are used in

queries. To provide a more accurate system, most schema matching systems combine multiple matching techniques.

Most schema matching solutions focus on one-to-one matches where one element in the source schema corresponds to another element in the mediated schema. However, matches in real world schemas can be more complex - such as many-to-one - where more than one source schema element corresponds to one element in the mediated schema. For example, name attribute in the mediated schema corresponds to the concatenation of firstName and lastName attributes in the source schema.

Figure 1-2 shows the architecture of a schema matching system (Doan et al., 2012).

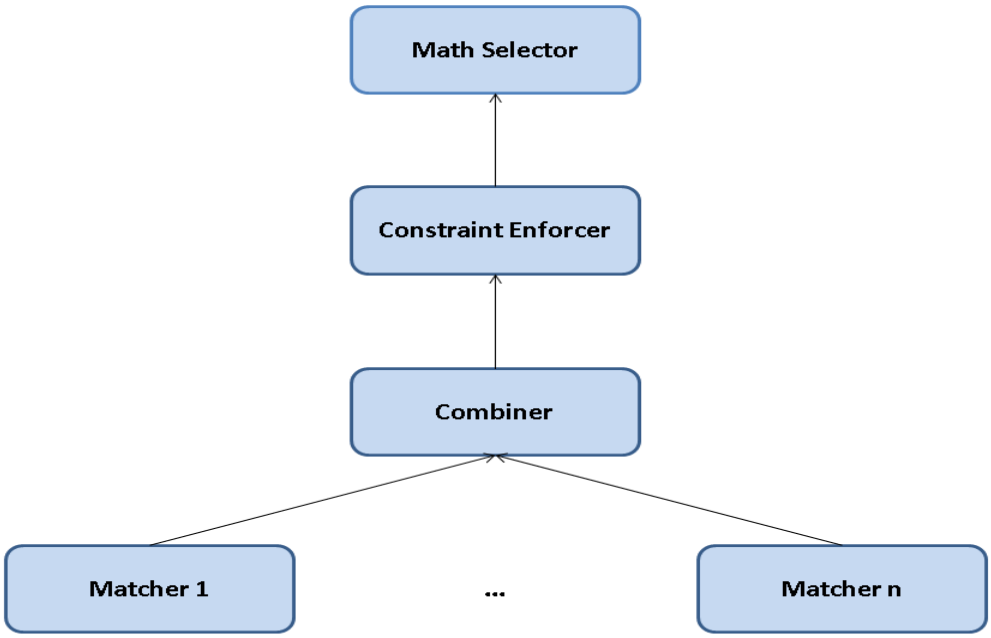

Figure 2-1: schema Matching System Architecture

Schema matching system components:

- Matchers: take two schemas, source schema S and mediated schema M to produce a similarity matrix. Each element in this matrix represents a pair-one from S and the other from M. These pairs will assigned a value between 0 and 1, were a higher value means the matcher is more confident that this match is correct.

- Combiner: merges all similarity matrices output from matchers to produce one similarity matrix. Combiner may use maximum, minimum, average or more complex techniques to merge these matrices.
- Constraints Enforcer: uses domain knowledge to prune the matches that do not convey domain constraints. It transforms the matrix produced by the combiner to another matrix that better reflects the domain constraints.
- Match Selector: produce the matches from the similarity matrix output from the constraint enforcer. It may use different techniques to do that, the simplest one is by selecting all pairs with similarity score that exceed particular value and return them as a match set.

Schema matching systems usually perform repetitive tasks to generate the matches, for example, comparing tens or hundreds of data sources against the mediated schema. Therefore some schema matching systems use machine learning techniques to reuse the previous matches and learn from them to produce new matches. LSD (Learning Source Descriptions) is one of the systems that uses

machine learning techniques in schema matching (Doan, Domingos, & Halevy, 2001).

The elaboration process from matches to mapping requires combining and converting matches to full coherent transformation expression, these expressions will be used for transforming data from the source schema to the target schema. Matches specify the relationship between schema elements but they do not specify which data value has a relationship with which data value and what operations need to be performed to transform data between schemas. In other words, matches specify SELECT and FROM clause, while mapping specifies the WHERE clause that relates to a schema elements value. Mapping also specifies any other functions that need to be applied on the data. In this way mapping transforms data between mediated schema and data sources or vice versa depending on the mapping language (LAV, GAV or GLAV). Different techniques for partial automatic schema matching are described in (Rahm & Bernstein, 2001).

In the early years, schema mapping was generated manually: designers wrote scripts that transform data from source schema to

target schema (mediated schema) or vice versa. To write such scripts, they should have a good understanding of the semantics for both schemas. As schemas get bigger, writing mapping manually becomes a harder process that takes a long time and requires a lot of effort since the knowledge of schema semantics is distributed on multiple people. Therefore all of them need to combine their knowledge to create the desired mapping.

Automatic or semi-automatic schema mapping systems become vital to produce mappings in large and complex schemas. Such systems take the set of matches constructed by the schema matcher and produce all possible mapping between source relations and target relations. Moreover, some of these systems provide ranking for the produced mapping and allow the designer to modify the ranking and choose the correct mapping.

Clio (Hernández, Miller, & Haas, 2001) , a tool developed by IBM, is one of the first automatic mapping tools, Clio employs a mapping-by-example approach by showing how a value from a target attribute can be generated from a value of source attribute. Clio takes two schemas and allows users to define the correspondence

in a visual interface and then generates the desired mapping as an SQL expression, while providing users with data samples and other feedback, to give them the chance to understand the produced mapping.

2.4 Summary

Building data integration systems begins with specifying the correspondences between attributes in the mediated schema and the attributes in the source schema. Then matches will be elaborated into mapping. Mapping provides a transformation expression that relates schema values in the different sources. Finding the matches and generating mapping manually using the designer's knowledge is not an efficient approach. While automating schema matching and mapping become one of the main challenge in data integration field.

3 Chapter3: Semantic Heterogeneity

Semantic heterogeneity occurs as a consequence of different interpretations for real-world concepts by designers in the process of conceptualization(George, 2005). This happens because designers may describe same real-world objects in different ways by using different terms to refer to the same concept (e.g., gender and sex), or they refer to different real-world objects using the same terms (e.g., court to refer the place where people playing sport and to refer a tribunal presided over by a judge or group of judges) because this term may give different meaning depending on the context.

As discussed earlier schema mapping is the main component in source description. To create a schema mapping, we should be able to reconcile semantics of data sources to produce the correspondences (matches). This is done in the early process of information integration, and these correspondences will be elaborated later into mapping.

3.1 Semantic Heterogeneity Reconciliation

Semantic reconciliation is important for information interoperability; it enables systems to communicate with each other and it is essential in data integration. Matchers need to create correspondences between attributes. Therefore, they need to reconcile the semantics of attributes.

Approaches used to reconcile data heterogeneity can be categorized as schema-based and attribute-based. Schema-based approaches use schema information to identify the semantic similarity in data. Furthermore, they focus on reconciling types before reconciling attributes. In contrast, attribute-based approaches use statistical and structural information of attributes to identify the semantic similarity of data in different sources (Parsons & Wand, 2003).

Manually, reconciling heterogeneity of data sources using designer's knowledge and data source documentation is not a practical solution. Designers might, for example, forget about the semantics of the data source, retire, or move to another company.

Furthermore, the documentation that they wrote might be out of date (Doan & Halevy, 2005).

Using clues discovered from schema information and data itself will not recover the full semantics of the data sources since these clues (e.g., data type for attributes, attributes name and integrity constraints...) only cover the representation and syntax side of the concepts. As a result, we need another type of clue that reflects the real semantics of entities and attributes in the data sources. Using domain ontology as a clue for the semantic reconciliation can provide higher level of information that can be used for semantic reconciliation. In this technique the schema of data sources will be mapped into the ontology and later matches will be constructed based on the relationships on that ontology. However, this approach may not be a practical solution in real life integration scenarios (Wang & Pottinger, 2008) because it is hard to specify the border of each domain and it is hard to agree on the standards since some organizations already establish their own schemas and follow their own standards.

As discussed before, a schema matcher is used to create correspondences between attributes while giving users the ability to refine matches on a visual interface. Generated correspondences from matchers are not expressive enough to represent semantics between data sources. Deepening mapping semantics by modeling the relationship between attributes more accurately produces richer application integration semantics (Wang & Pottinger, 2008). This can be done by modeling the relationship between attributes more accurately, for example, schema matching may discover that "Juice" in one schema equals to "Beverage" in another one. However, more accurate modeling is by saying that "Juice" is specialization of "Beverage". SeMap (Wang & Pottinger, 2008) is a system that construct richer mapping by taking into account the following relationship types: 'Has-a', 'Is-a', 'Associates' and 'Equivalent'.

In existing schema mapping techniques, the semantic reconciliation between data sources is performed by matchers, which find the correspondences between attributes of similar or the same meaning. In contrast, attribute-based semantic reconciliation of multiple data sources (Parsons & Wand, 2003) is an approach for

semantic reconciliation based on the relations between properties instead of the similarity in meaning. This approach uses ontology to deal with semantic heterogeneity, based on the fact that databases store information that describes things and concepts. Ontology is a field interested in describing what 'is' in the real world, in particular, things and their properties. More specifically Parsons and Wand (2003) use the notion of property precedence in Bunge's ontology (Bunge, 1977) to identify the semantics of properties.

3.2 Property Precedence:

Parsons and Wand argue that the problem of semantic reconciliation can be reduced by addressing the semantics of attributes independently from any other higher-level constructs such as class or entity type (Parsons & Wand, 2003). The concept of property precedence proposed by Parsons and Wand for semantic reconciliation relaxes the assumption of inherent classification, the idea that database instances explicitly or implicitly belongs to classes or entity types (Parsons & Wand, 2000).

Anchoring instances with classes or entity types cause data integration to be more complicated. A solution for this problem was proposed in (Parsons & Wand, 2000), which recommends the separation of representing instances and properties from any specific classification. More specifically, they defined classes as a second layer above instances and their properties. Classes are defined as certain set of properties and the meaning of a class are these instances that possess the specified properties. Parsons and Wand believe that the semantics of data can be captured at attribute level.

Bunge's ontology (Bunge, 1977) describes the semantics of a property according to the relationships with other properties. Parsons and Wand use Bunge's ontology because they want their approach to be general and independent of any domain. Bunge's ontology deal with systems in general and it is comprehensive ontology that includes many other ontologies.

One of the main relationships in Bunge's ontology is property precedence. Parsons & Wand (2003) use the notion of property precedence as a basis to reconcile the heterogeneity of attributes.

The key to their approach is that “attributes that appear different may represent the same concept at a more generic level” (Parsons & Wand, 2003). For example, in a database that describes characteristics of people, a person who has the attribute “Having black hair” and the person who has the attribute “Having brown hair”, both have hair. Since having a color for their hair implies that these people are not bald. In other words, we can say that, the attribute “having hair” precede both attributes “having black hair” and “having brown hair”.

Property precedence has been defined as following (Parsons & Wand 2003):

Let P1 and P2 two designate properties. P1 said to **precede** property P2, if every instance X possessing property P2 possesses P1 as well. In other words, if the set of instances that possess P2 are a subset of the instances that possess P1, then P1 precedes P2. It is also equal to say that p2 is preceded by p1 (P2 Preceded by(P1)). We will refer to this relation between properties as a subsumption relation, which will be the basic for building the

Property Precedency Graph (PPG). For example, attribute “having eyes” precedes attribute “having green eyes”.

Using property precedence we can infer implicit properties from explicit properties. For example, the property “having two legs” and the property “having legs”, as long as a thing is possessing the property “has two legs” it is also correct to say that this thing “has legs” as well. Possessing the explicit property “has two legs” allow us to infer the implicit property “has legs”.

The following are definitions for concepts that will be used in this thesis:

Definition 3.1: The scope of a property P is the set of all instances that possess the property P. Denoted by Scope (P). *

According to definition 3.1, precedence can be defined as following: Property P1 precedes property P2, if and only if the Scope(P2) ⊆ Scope(P1). It is also valid to say that property P2 is preceded by property P1 (P2 Preceded by(P1)).

Definition 3.2: The form of an instance X is the set of all properties possessed by X. Denoted by Form(X).*

***Definition 3.3:** A manifestation of a property P in an instance X, is a property of X preceded by P. Denoted by Manifest(P,X) = Form(X) ∩ Preceded by (P) .*

***Definition 3.4:** The property P is said to be fully manifested by a set of preceded properties S if everything possess the property P possess at least one of the properties in S. **

Manifestation of a generic property can have two types. First, generic property manifested using a specific value. For example, the generic property “age” can be manifested by having a specific age X. In this type of manifestation there is no meaning for the property value without attaching it to the generic property. Second, generic property manifested using a specialization of the generic property,.For example, the generic property “move” can be manifested as “swim”, ”run”, ”walk” or “crawls”. Notice that unlike the previous type of manifestation, the values that manifesting the

generic property have meaning independently without attaching it to the generic value “move”.

3.3 Use Precedence and Manifestation to Identify Semantic Similarities

Parsons and Wand (2003) use the notion of property precedence and manifestation to create a new definition for properties similarity. According to this definition, property P1 is similar to property P2 if and only if they are a manifestation of the same higher level property P. In other words, property P1 and P2 are similar if both are preceded by property P.

This definition provides a new approach for semantic reconciliation and data integration, showing that attributes that appear different may be manifested by the same higher-level property. For example, finding all grad students in a university database can be accomplished a query that searches for all students who registered in a grade course, or by a query that searches for all students with a supervisor. Although registering in a grad course and having a supervisor are not similar properties both can be manifestations of

the same higher level property “Being a grad student”. This example shows how the similarities of properties are inferred if they are manifestation of the same general property (i.e., preceded by the same general property), even if the properties are from different domain and have different meaning.

To use this approach for data integration, we need to find a way to identify the interior precedence (precedence relations between properties in a data source), the exterior precedence (precedence relations between properties across different data sources) and the common generic properties of properties in different sources. The properties and the precedence relation between properties in the data source will form a directed non-connected graph. We will refer to this graph as a Property Precedence Graph (PPG). It is non-connected because not all properties are guaranteed to have precedence relationship between them. As a result, we will have some nodes (properties) without a path between them.

PPG= (V, E):

- V is a set of attributes called vertices or nodes.

- E is set of ordered pairs of vertices, called arcs or edges. Having the pair (V1, V2) in E means that the attribute V1 precedes the attribute V2. In other words there is a path from V1 to V2.

Definition 3.5: A generic property P is **fully manifested** by a set of properties S, if everything that possesses P possesses at least one of the properties in S. In other words, if the union of all property scopes in S equals the scope of P. *

Two Lemmas can be used to establish relationship between properties based on the precedence and manifestation (Parsons & Wand, 2003).

Lemma 1: Let P1 and P2 two generic properties, and S1 and S2 set of preceded properties for P1 and P2, respectively. P1 precedes P2 if P2 is fully manifested by S2 and every property in S2 is preceded by a property in S1. *

This lemma enables us to infer the precedency relation between generic properties based on the relationship between their manifestations. For example, suppose the property “age category”

(P1) manifested by S1= {“child”, “teenage”, “adult”, “middle age”, “old”}, and the property “age” (P2) is fully manifested by age values {5, 16, 25, 38, 60}, there is already a known mapping between manifestation for S2 and S1, since every age can be mapped to one of the age group values then the property “age category” precedes the property “age”.

Lemma 2: Let P1 and P2 two generic properties, S1 and S2 set of preceded properties for P1 and P2, respectively. Each property in S2 is preceded by one or more properties in S1, if P2 is preceded by P1 and P1 is fully manifested by S1[1]. *

This lemma enables us to infer the precedence relation between the manifestations based on the relationship of their generic properties. For example, suppose that the property “being employee” (P1) is fully manifested by S1= {“full time”, “part time”}, and the property “job role” (P2) is manifested by S2= {“programmer”, “database administrator”, “analyst”, “designer”}, in order to have a job role you should be an employee (the

[1] * Taken from Parsons and Wand(2003)

property “job role” preceded by the property “being employee”), by applying lemma 2, the manifestation of the property “job role” preceded by one or more properties in the manifestation of the property “being employee”.

The previous two lemmas enable us to establish semantic relations between properties across sources. Furthermore, the first lemma enables us to identify generic properties across several sources that are not defined within data sources.

We notice that using property precedence for semantic reconciliation enables us to infer implicit properties from explicit properties. For example, for gender specific diseases we can infer the gender of the patient depending on his disease. For example, one can infer that a patient is female if the patient is suffering from cervical cancer. Such information from property precedence enables us to produce enhanced correspondences between attributes and, as a result, richer mapping between data sources. Other researches consider limited type of relations between attributes, such as, hypernyms/hyponyms relation that are

considered in (Do, H. H., & Rahm, 2007). However, property precedence covers such relation between properties.

3.4 Conclusion

Semantic heterogeneity is one of the main obstacles in building data integration systems. Reconciling semantic heterogeneity can be accomplished using matchers; matchers produce correspondences between similar or same attributes. However, these correspondences may produce poor mapping between sources because they do not model the relationship between attribute in accurate way. Modeling the relationship between attributes more accurately will produce richer mapping. Parsons and Wand (2003) proposed an approach for semantic reconciliation based on the relations between properties instead of the similarity in meaning. They use the notion of property precedence as described in Bunge's Ontology. In order to use this approach for data integration, we need to find a way to identify the precedency between attributes in and across data sources and common generic properties of properties in different sources. The precedence relations between

attributes will form non-connected directed graph, which we refer to as a Property Precedence Graph (PPG).

4 Chapter4: Use Property Precedence to Support Data Integration

In the previous chapter, we show how to discover the semantics between attributes using the notion of property precedence and manifestation. In this chapter we will show how to identify all local precedence relations between attributes in data source, how to identify the generalized properties that can be used to bind multiple data sources and how to use the precedence relations in reformulating queries.

4.1 Build Property Precedence Graph

To be able to use property precedence in data integration we need to identify the local precedence between properties within data sources (these precedencies will form the PPG) and the global

precedence relations between attributes across data sources. Global precedence relations will be used to bind data sources. We propose two algorithms to build two types of local PPG for each data source.

The precedence relations between properties in a data source will form a property precedence graph that can be used to reconcile heterogeneity at schema level, we will call such a graph a, property precedence graph for schema (PPGs). In contrast, the precedence relations between the manifestations of properties will form property precedence graph that can be used to reconcile heterogeneity at data level, we will call such a graph a, property precedence graph for data (PPGd).

We use a bottom-up approach to build both PPGs and PPGd by using data itself. Consequently, although the precedence relations are correct at the time they are discovered, they may disappear when more instances are known. Moreover, although data in a single field of a database may represent several facts (Properties), our approach can only create relations between single properties. As a

result, data should be transformed such that each field represents a single fact before the PPG is constructed.

Our approach for building PPGs is based on subsumption scopes of properties without taking into account the manifestation of these properties. Algorithm I shows how to get properties scope (Scope of a property is the set of all instances that possess this property) in schema S.

Algorithm I: Find Scopes for Schema Properties

Input: Set of all properties P= $\{p_1, p_2, p_3 \ldots p_i\}$ in schema S

Set of all instances I = $\{i_1, i_2, i_3 \ldots i_j\}$ in schema S

Output: List of scopes, a scope for each property p, Scopes= $[scope_1, scope_2, scope_3 \ldots scope_i]$ and each scope will be represented as a set of instances.

Begin:

Scopes$\longleftarrow$[ ] of size i

For each property p_j in P **do** {

$Scope_j \longleftarrow$ { }

For each instance x in I **do** {

If x possesses property p_j **then**

$Scope_j \longleftarrow Scope_j \cup x$ }

Insert $Scope_j$ in Scopes list at index j }

Return Scopes;

After identifying the scopes of all properties in the schema, these scopes can be used in Algorithm II to discover all precedence relations between properties. Algorithm II shows how to extract all precedence relations between properties in schema S.

Algorithm II: Find Property Precedence Relations for Schema Properties (PPGs)

Input: Set of all properties P= {p_1, p_2, p_3…p_i} in schema S

List of all Scopes, Scopes = [$scope_1$, $scope_2$, $scope_3$ … $scope_i$] and each scope will be represented as a set of instances.

Output: Set of order pairs of properties R= {r_1, r_2, r_3 … r_j} for schema S, where r is in form of r= (p_x , p_y), meaning property p_x precedes property p_y.

Begin:

R= { }

For each property p_k in P **do** {

$Scope_k$ =get the scope of p_k from scopes list at index k

For each property p_z in p_{k+1} **do** {

$Scope_z$ =get the scope of p_z from scopes list at index z

If $Scope_k$.size >= $Scope_z$.size **do**

Intersection= $Scope_k$ ∩ $Scope_z$

If intersection = $Scope_z$ **do**

R= R ∪ (P_k , P_z)

If $Scope_k$.size < $Scope_z$.size **do**

Intersection= $Scope_k$ ∩ $Scope_z$

If intersection = $Scope_k$ **do**

R= R ∪ (P_z , P_k) } }

Return R;

The set of all precedence relations between properties R can be used to form the PPGs, where the vertices of PPGs will be the set of all properties P in schema S and the edges between vertices will be according to the pairs in the set R, where there is a directed edge between p_x and p_y if there exist a pair (p_x,p_y) in R. Note that PPGs is directed graph, the pair (p_x,p_y) means that the property p_x precedes the property p_y but not vice versa.

Algorithm III shows how to identify the scopes of properties manifestations. The scope for each manifestation will take into account the property that has this manifestation since some manifestations may have different meaning depending on what property they attached to. Moreover, other manifestations do not have any meaning without attaching them to their property, for example, numerical values do not have any meaning without attaching them to a property. As a result we may find more than one scope for each manifestation if there is more than one property with the same manifestation. Identifying manifestation scope will be based on both the property that has this manifestation and the manifestation itself.

Algorithm III: Find the Scope of Schema Properties and its Manifestations

Input: Set of all properties P= {p_1, p_2, p_3…p_i} in schema S.

Set of all manifestations for each property p, M={m_1, m_2, m_3…m_i }, where m is in form of, m_k=(v_1, v_2, v_3…) the manifestation for property p_k.

Set of all instances I = {i_1, i_2, i_3 … i_j} in schema S

Output: List of scopes, a scope for each manifestation value v of a property p, Scopes= [$scope_1$, $scope_2$, $scope_3$ …] and each scope will be identified by two keys, the property p and the value v of the manifestation.

Begin:

Scopes⟵[ ]

For each property p_j in P **do** {

m_j⟵List all values for p

For each value v_x in m_j **do** {

$Scope_x$= { }

For each instance i in I **do** {

If instance i possesses p_j and p_j has v_x in the manifestation set do

$Scope_x$ ⟵ $Scope_x$ U i }

Insert $Scope_x$ in Scopes list with two keys to identify that scope the property p_j and the value v_x } }

Return Scopes;

After identifying the scopes of all manifestations in the schema, these scopes can be used in Algorithm IV to discover all precedence relations between manifestations. Algorithm IV shows how to extract all precedence relations between manifestations in schema S, while taking into account which property is having that manifestation.

Algorithm IV: Find Property Precedence Relations for Schema Manifestations (PPGd)

Input: Set of all properties P= {p_1, p_2, p_3…p_i} in schema S.

List of all manifestation scopes, Scopes = [$scope_1$, $scope_2$, $scope_3$ …], and each scope will be represented as a set of instances.

We can identify a particular scope by using the property and the manifestation as a key.

Output: Set of precedence relations R= {r_1, r_2, r_3 … r_j} for schema S, where r is in form of r= (p_a:v_b , p_c:v_d), means value v_b precedes the value v_d when v_b is manifestation of p_a and v_d is a manifestation of p_c.

Begin:

R= { }

For each scope $scope_k$ in Scopes **do** {

For each scope $scope_z$ in $Scopes_{k+1}$ **do** {

If $scope_k$.size >= $scope_z$.size **do**

Intersection= $scope_k$ ∩ $scope_z$

If intersection = $scope_z$ **do**

$R = R \cup (p_k:v_k , p_z:v_z)$

If $Scope_k.size < Scope_z.size$ **do**

$Intersection = Scope_k \cap Scope_z$

If intersection = $Scope_k$ **do**

$R = R \cup (p_z:v_z , p_k:v_k) \} \}$

Return R;

The set of all precedence relations between manifestations in R can be used to form the PPGd, where the set of pairs of properties and their manifestation will form the vertices of PPGd and the edges between vertices will be according to the pairs in the set R, where there is a directed edge between the vertex $p_x:v_x$ and $p_y:v_y$ if there exist a pair ($p_x:v_x$, $p_y:v_y$) in R. Note that PPGs is directed graph, which means the directed path ($p_x:v_x$, $p_y:v_y$) means that the manifestation v_x precedes the manifestation v_y when v_x is a manifestation for p_x and v_y is a manifestation for p_y but not vice versa.

Updating the database might affect the scope of properties and/or the scope of property manifestations. To avoid re-running algorithms I and/or III to re-calculate the scopes, storing and

tracking each property scope and each “Property value” scope will solve this problem. For example, when the database update causes an instance to possess a new property, the instance will be added to that property scope, If the database update causes an instance to possess a new property and stop possessing an old property, the instance will be removed from the old property scope and added to the new property scope.

As a result of scope changes, the precedence relations might change as well. To avoid re-running algorithm II and/or IV and guarantee that the PPG reflects the status of data, we can make partial updates for PPG as follows: remove all the edges (incoming and outgoing) between the property that has new scope (Property x) and the rest of the properties that are connected to this property, re-intersect the new scope for property x with the rest of the properties updated the property precedence relations. These relations will connect property x with the rest of the properties in the graph.

We implemented Algorithms I-IV using Jena library and demonstrate the running of these algorithms on a data set from DBpedia. The

result is a set of precedence relations that can be used to form PPGs and partial precedence relations in PPGd (partial because there is large number of scopes for manifestations that need to be compared) We will return to the implementation and results in chapter 5.

4.2 Use Property Precedence Graph in Creating Common Generic View Over Multiple Data Sources

After building PPGs and/or PPGd for data sources using the algorithms in section 4.1 (PPGd if we want to reconcile heterogeneity at data and schema level and PPGs if we want to reconcile heterogeneity at schema level only), data sources can be joined using generalized properties that define common semantics across data sources.

We can define a generalization of properties using property precedence as following:

Definition: A generalization property G for a set of properties $\{p_1, p_2, p_3 \ldots.\}$ is a property that precedes all the properties in that set.

If there is no explicit common generalized properties that can be used to bind two data sources, we can identify the generalized property for properties in two separated data sources using Lemma1 as described in section 3.3. However, Lemma 1 can be used if data sources contain different level of data granularity; one of the data sources contains more specific property than the other, and if there is a known mapping between the manifestations of the generic properties in both sources. In this way Lemma 1 can be used to infer implicit properties in the source that contains more specific properties, from explicit properties in the source that contains more generalized properties.

In our approach, we will use the generalized properties to bind data sources. For example, suppose we have two data sources that provide information about students in a university. In source 1 the property “being student” precedes both properties “graduate student” and “undergraduate student” and in source 2 the property “program of study” precedes a list of academic programs. If there is a known mapping between the manifestation of the property “being student” in source 1 and the manifestation of the property “program of study” in source 2 (some programs are only offered for

graduate students like MPA or only for undergrad students like Bachelor of science) then, according to Lemma 1, we can infer that the property "being student" precedes the property "program of study". The property "being student" will be consider now as a generalized property that can be used to bind these data sources. This scenario is described in Figure 4-1

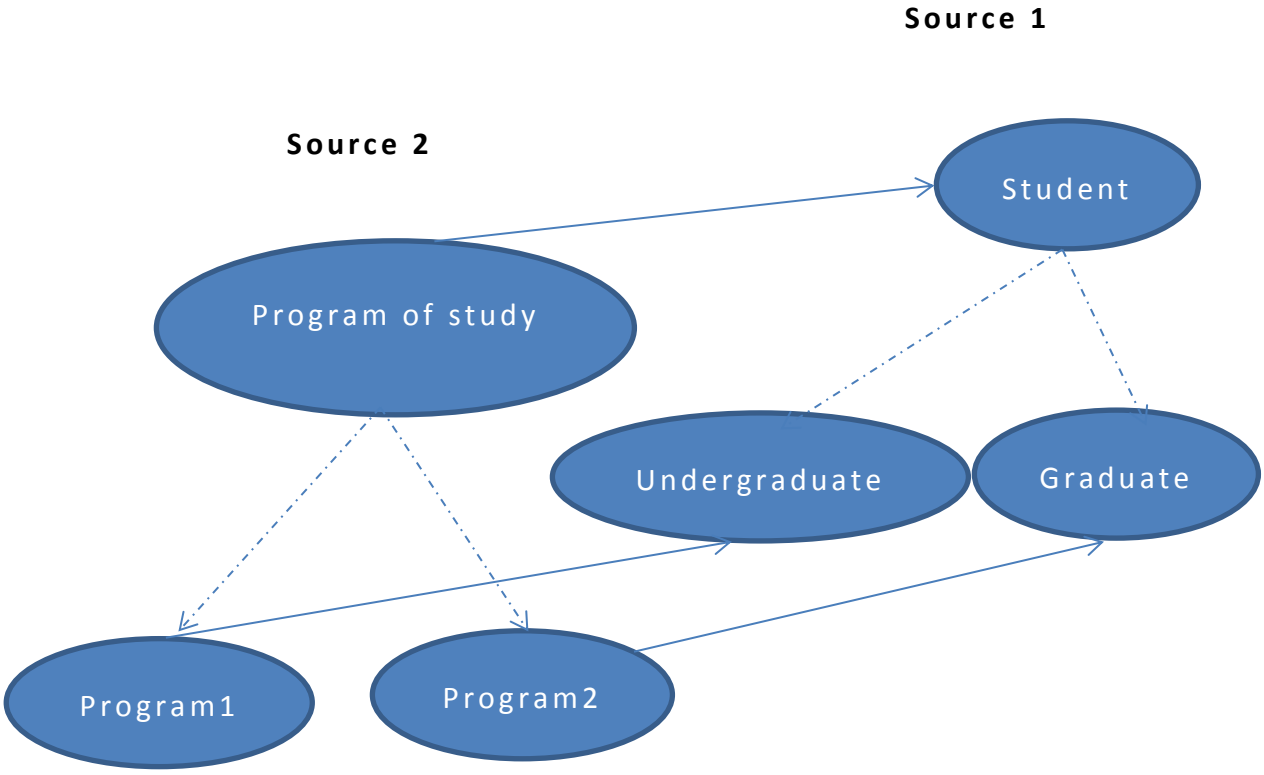

Figure 4-1: Generalized property example (dashed lines represent manifestation relation and the solid lines represent a precedence relation)

Other techniques can used to identify the generalized property If the two conditions that we mentioned before do not hold to use Lemma1. For example, it may be possible to map the PPG for both data sources to an ontology and identify the generic properties that hold common semantics for both data sources using that ontology.

The common generalized properties can be used to form a generic view over multiple data sources. This view can be used to pose queries against these sources and the resulting property precedence relation between the generalized property and the local property in the data source “Global precedence relations” will be used in query rewriting along with the Local precedence relations.

4.3 Use Property Precedence Graph in Rewriting Queries

Query rewriting is defined as the following problem: for a query q find a query q` or set of queries such that the answer of q over the source schema returns the same result as the answer of q` over the target schema (Arenas 2004).The basic query rewriting algorithm

consists of four main stages, rule generation, query translation, query optimization and query assembly (Bellahsene, 2011)

Relying on property correspondences is not enough to handle all the semantic heterogeneity in rewriting queries. Use PPGs or PPGd along with the generalized properties in rewriting queries will provide opportunities to obtain more potential answers.

Sekhavat (2014) propose two algorithms to rewrite queries using the PPG and generalized properties, each of them use different query expansion technique (expansion through generalization and expansion through specialization) resulting a data integration system that can be configured according to the user preference. Users can choose the query rewrite type based on their preference for sound (complete) or accurate answers.

If the user prefers to get accurate answers then the query rewriting will be by replacing generalized property (preceding property in the precedence relation) in the target query with a more specific property (preceded property in the precedence relation) in the source schema to retrieve data that match this specific property.

On the other hand, if the user prefers to get a complete answer then the query rewriting will be by replacing the specific property (preceded property in the precedence relation) in the target query with a more general property (preceding property in the precedence relation) in the source schema to retrieve correct data as well as some unwanted data.

For example, if a target query asks for name of patients who suffer from Acne aestivalis, this can be rewritten using expansion through generalization by querying all the patients in the source schema who suffer from cutaneous condition (there is a known precedence relation that cutaneous condition precedes Acne aestivalis). In this way we retrieve all patients who suffer from Acne aestivalis and other patients who might suffer from Acne aestivalis while other techniques might fail to retrieve some patients if they do not explicitly possess the attribute Acne aestivalis.

4.4 Conclusion:

In order to use property precedence approach in data integration we need to identify local precedence relations in data sources and

global precedence relations between data sources. We use a bottom up approach to identify local precedence in a data source using subsumption scope of attributes and data. We propose and implement four algorithms to build two type of PPG, each which can be used to reconcile heterogeneity of data in different level (PPGs for schema level and PPGd for data level).

To identify the generalized properties that can be used to bind two data sources, Lemma1 can be used if data sources contain different level of data granularity and if there is a known mapping between the manifestations of the generic properties in both sources. The generalized properties can serve as a common view for users to pose their queries.

Rewriting queries using PPG allow users to retrieve answers based on their preference of completeness and accuracy. Replacing a property in the target query with the preceding property will retrieve more potential answers (completeness) while replacing property in the target query with the preceded property will retrieve data that will match the preceded property (accuracy).

5 Experiment to build PPG on a Dataset from DBpedia

In the previous chapter we proposed algorithms to build PPG (PPGd and PPGs) using scope of attributes and manifestations. We also explain how to use the precedence relations in rewriting queries and in creating a common view over multiple data sources. In this chapter we discuss the implementation of the previous algorithms on a dataset from DBpedia and we will show our results with examples on how to rewrite query using the extracted precedencies.

5.1 DBpedia

DBpedia is a crowd-sourced community that aimes to extract structured data from Wikipedia and make it available in the web. The extracted structured data is available in machine readable format “Resource Description Format” allowing semantic web techniques to be employed against this data source, such as, linking DBpedia with other open linked dataset, asking sophisticated

questions against DBpedia and building applications based on the DBpedia contents (Auer, 2007).

The DBpedia dataset is interlinked with many other open linked data sources using RDF links. Figure 5-1 contains an image taken from the linkeddata.org website showing how DBpedia serves as a hub for open linked data sources in the semantic web. This figure been added to show the scope of open linked data.

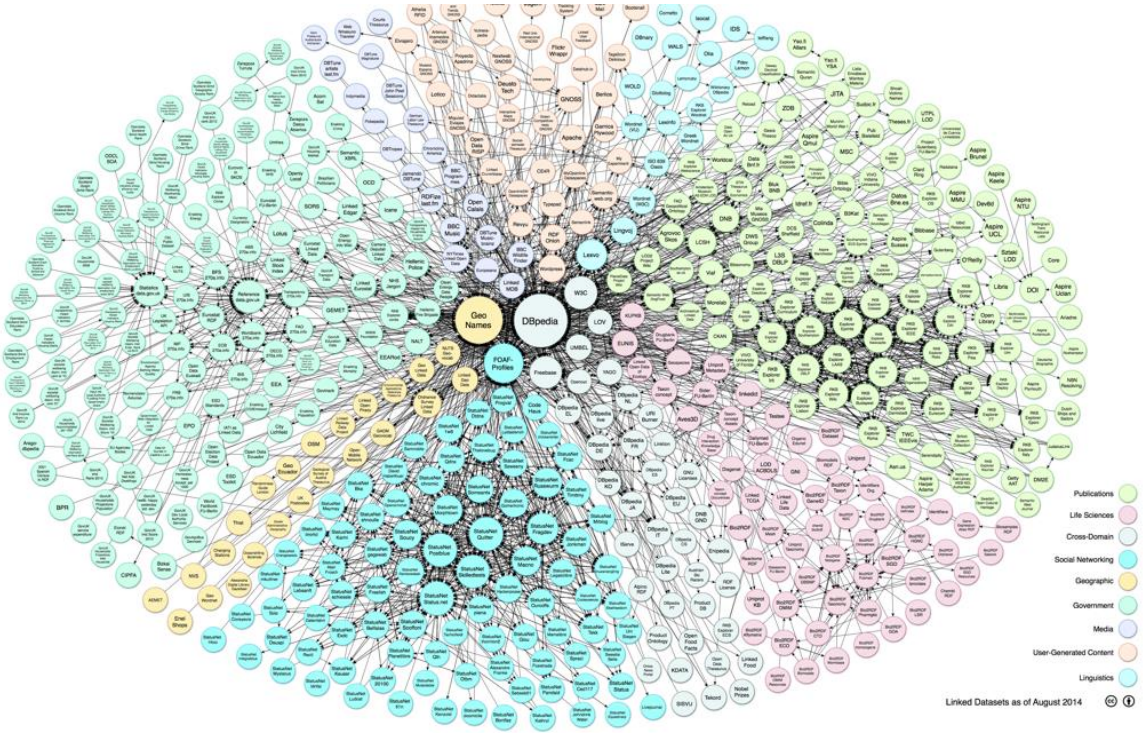

Figure 5-1 Linking Open Data cloud diagram 2014, by Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/

Resource Description Framework (RDF) represents data as triples of subject, predicate and objects, each of these will be represented using uniform resource identifier (URI) which is unique for each concept. A triple will represent a specific fact about the subject, for example, the following triple represent a fact about Canada's population (Canada, population, 35 million).

Each triple can be linked with other triples using predicates that link the object of the first triple with the subject of the second triple. For example, taking the triple (Stephen Harper, prime minister, Canada) and the triple (Canada, capital, Ottawa) are linked together since the object of the first triple is the subject of the second triple. This can be possible because Canada is represented using the same URI in both triples.

DBpedia consists of multiple RDF dataset that represent different parts of the Wikipedia page, Title dataset, short abstract dataset, long abstract dataset, geographic dataset, weblinks dataset, info box dataset and categories dataset.

5.2 Experiment and results

We choose DBpedia as an experiment data source to demonstrate that our property precedence and our approach in extracting the precedency is not limited to one domain since DBpedia is a cross domain data source.

In our experiment, we combine the triples (records or facts) of three different DBpedia datasets, title dataset, categories dataset and info box dataset to apply our algorithms and to create the PPG. These datasets were chosen because data in these sets are normalized which make it possible to apply our algorithms without any prior processing for data. The resulted dataset contains 41931660 triple that describes almost 4 million different concepts using 1374 predicates (property).

We use Jena API (free open source Java library to build semantic web and linked data applications) to implement our algorithms and to manipulate the RDF triples in DBpedia. TDB (triple data store) is used to host and store DBpedia dataset locally rather than querying DBpedia online.

First we build PPGs by running algorithm I and algorithm II, we use algorithm I to identify the set of all instances that possess each predicate (scope) and the output of this algorithm used as an input for algorithm II. Algorithm II identifies the precedence relations between predicates by intersecting all predicates scope with each other to identify the subsumption relation between them.

After eliminating the 38 cases of equal scopes (e.g., http://dbpedia.org/ontology/lastFlightStartDate and http://dbpedia.org/ontology/fuelCapacity) from our results since there is no meaning for predicates with equal scope in property precedence (if two predicates have the same scope this does not mean that these predicates are similar) we got 2019 precedence relation.

Then we build partial PPGd by running algorithm III and IV. We could not identify all the precedence relations in this case since we have huge number of scopes that need to be compared. Each attribute will have multiple values and each value associated with the attribute will have its own scope. The number of precedence relations discovered after running algorithm IV for one week was

542. We write our results in JSON file as a source and target nodes in order to visualize them using D3 library, the following is a sample of the JSON file:

Figure 5-2: Sample of the precedence relations extracted by algorithm II

The following figure shows partial PPGs build from algorithm II, nodes in this graph represent the attributes and the direct edges represent the precedence relation between attributes. Notice that most of the attributes are connected together with multiple edge (i.e., they precede or are preceded by other attributes)

Figure 5-3 Subgraph from PPGs build from DBpedia

The following is a smaller sub graph of the PPGs showed in figure 5-3.

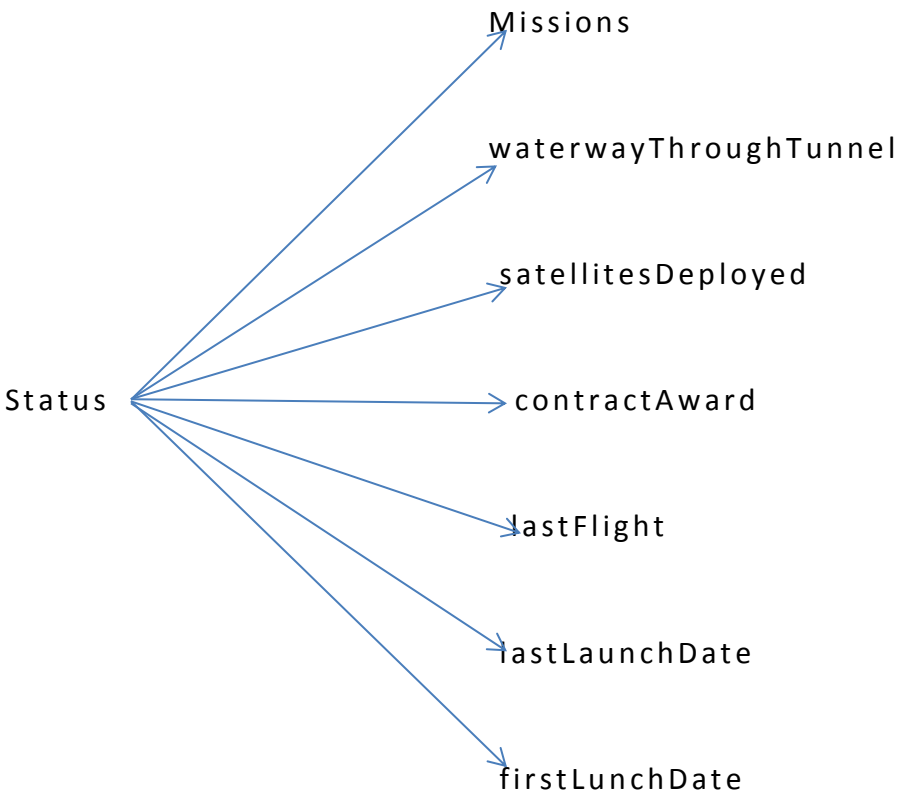

Figure 5-4: Subgraph from PPGs

This graph can be used to rewrite a target query that asks for the status of a particular mission using expansion though generalization technique that we explain in chapter 4, the property mission in the target query will be replaced by status property in the source, this will retrieve complete results of instances that possess mission status as well as instances that do not explicitly possess mission status.

The following table shows a comparison between number of unique instances retrieved when executing a query using expansion through generalization and without using the expansion technique. Query rewriting using expansion through generalization is done by replacing the property in the *Where* clause with a more generic property (preceding property).

Where Clause Property	**Expansion through generalization**	**Without expansion**
missions	37254	539

bowlRecord	5481	533
Port2Docked Time	82	1

Table 5-1 query expansion through generalization

We notice the increase in recalls while we are sure that we did not lose the precision in the resulted set since the instances that possess the generalized property will include the instances that possess the specific property.

Another subgraph is represented by the following relations:

```
{source: "http://dbpedia.org/ontology/language", target: "http://dbpedia.org/ontology/officialLanguage", type:"suit" },
{source: "http://dbpedia.org/ontology/language", target: "http://dbpedia.org/ontology/regionalLanguage", type:"suit" },
```

A target query that has language in the queried attributes can be reformulated using expansion through specification to get accurate results about the official language by replacing the attribute language with the attribute officialLanguage.

The following table shows a comparison between number of instances retrieved when executing a query using expansion through specialization and without using the expansion technique. Query rewriting using expansion through specialization (when general property has one or more specific property) is done by replacing the property in the *Where* clause with a more specific property (preceded property).

Attribute	Expansion through specialization	Without expansion
language	201	72858
Capital	2	3016
College	4	12895

Table 5:2 Table 5-1 query expansion through specialization

We notice that there is less recalls while the precision is increased because the resulted set will only have the instances that possess that specific property.

As we show, the constructed graph for each data source can be used as metadata for query rewriting and for reconciling the semantics of properties inside each data source. Furthermore, by combining these graphs using generalized properties we can form a bigger graph that can be used to reconcile the semantics of attributes between different data sources.

The following relations were constructed using algorithm IV, it represent some precedence relations for values of the property LastFlightStartDate.

Value http://dbpedia.org/resource/Soviet_Union of Predicate http://dbpedia.org/ontology/countryPrecede
Precede
Value 1989-12-29^^http://www.w3.org/2001/XMLSchema#date of Predicate http://dbpedia.org/ontology/lastFlightStartDate

Value http://dbpedia.org/resource/Category:Space_Shuttle_Challenger_disaster of Predicate http://purl.org/dc/terms/subjectPrecede
Precede
Value 1986-01-28^^http://www.w3.org/2001/XMLSchema#date of Predicate http://dbpedia.org/ontology/lastFlightStartDate

Value http://dbpedia.org/resource/Category:Manned_spacecraft of Predicate http://purl.org/dc/terms/subjectPrecede
Precede
Value 1986-01-28^^http://www.w3.org/2001/XMLSchema#date of Predicate http://dbpedia.org/ontology/lastFlightStartDate

Value http://dbpedia.org/resource/Category:1986_disasters of Predicate http://purl.org/dc/terms/subjectPrecede
Precede
Value 1986-01-28^^http://www.w3.org/2001/XMLSchema#date of Predicate http://dbpedia.org/ontology/lastFlightStartDate

Value http://dbpedia.org/resource/Category:Space_Shuttle_orbiters of Predicate http://purl.org/dc/terms/subjectPrecede
Precede
Value 1986-01-28^^http://www.w3.org/2001/XMLSchema#date of Predicate http://dbpedia.org/ontology/lastFlightStartDate

Value http://dbpedia.org/resource/Category:Destroyed_spacecraft of Predicate http://purl.org/dc/terms/subjectPrecede
Precede
Value 1986-01-28^^http://www.w3.org/2001/XMLSchema#date of Predicate http://dbpedia.org/ontology/lastFlightStartDate

We notice two important things; first, the values that precede LastFlightStartDate values are from total different domain, lastFlightStartDate values point to specific dates and other properties value are strings without any date relation except one case “1986_disaters”. Second, in each relation the property name (country_precede and subject_precede) is totally different from LastFlightStartDate. Despite the differences in value domain and in attributes name our approach was able to construct a relation (match) between these values.

To prove that the precedence relations mentioned above is not a fake relations I investigate the first precedence relation pair (Soviet_Union, 1989-12-29), if we search in the history we find that In August,1968 Warsaw Pact (headquarter in Soviet Union “Moscow”) invade Czechoslovakia, 1989-12-28 was the end of velvet revolution which ends the communist rule in Czechoslovakia after the Warsaw Pact invasion and cause the conversion to a parliamentary republic. Despite the fact that 1989-12-29 and Soviet_Union are not directly related but our technique was able to find a relation between these two properties.

Using PPG for semantic reconciliation provides advantages over other techniques discussed in the literature review:

- It provides definite relations (matches) between properties. If one property precedes another property this means that there is a definite match between these two properties even if they have different names and different domain of values (the two properties represent the same concept at different level of abstraction). In contrast other techniques that we discuss

before give a rate for each match which indicates how accurate this match is,

- It provides matches that other techniques cannot discover. Matches in PPG do not require similarity in properties name or domain for property values.
- Query rewriting using property precedence graph can provide potential answers that other techniques cannot retrieve since using expansion through generalization might retrieve answers without even requiring the instances to possess the searched property/value. This technique will increase false positive results but we get a complete set of answers that other techniques cannot provide.

5.3 Conclusion:

In this chapter we show the results of implementing the algorithms developed in chapter 4 on DBpedia data source with a few examples on how to reformulate queries using both techniques expansion though specialization and expansion through generalization. Advantages of the constructed PPG were discussed as well.

6 Conclusion and future work

In this thesis we introduce and implement algorithms to build property precedence graph. We implement these algorithms on a dataset from DBpedia that contains 42 million triple with a size of 7 gigabyte.

Property precedence graph can be used as a metadata for semantic reconciliation in data integration system, so rather than trying to find matches between attributes in different data sources using different type of matchers, this graph can be used to lookup matches between attributes inside each data source and between attributes in different data sources.

Matches provided by property precedence graph are accurate and definite so if there is a precedence relation between two attributes we are confident that these attributes are similar, in contrast with the matches provided by other techniques might be wrong because they are based on linguistic and statistics clues. Such techniques require a human to confirm such match. As a result most data

integration systems use more than one type of matcher to combine the results of each match and get more accurate matches.

We also explain how to use expansion through generalization or expansion through specialization to rewrite queries using PPG, this way of query rewriting will guarantee answers that cannot be retrieved by other rewriting techniques, since in this type of rewriting technique instances do not have to explicitly possess properties to be in the retrieved result set.

The concepts discussed in this thesis (property precedence, property precedence graph, generalized property) can be used as the basis of a new data integration architecture. In this architecture a PPG will be created for each data source and these data sources will be combined using generalized. As a result we will have a bigger graph for rewriting queries, the generalized properties in this graph will serve as a mediated schema for users to pose query against.

This thesis focuses on how to extract the property precedence relation from one data source and the data in this data source

should be complete and normalized. Further research is required to identify precedence relations if the dataset is not complete and future experiments should include more than one data source as an effort to combine the resulted PPG to one large PPG.

In this thesis we only have one method to extract the generalized property (Chapter 4 Lemma 2). Further research is required on approaches that can be used to identify the generalized properties between multiple data sources. It may be feasible to use the relations in online knowledge base like Freebase to solve this problem. Enhancing query rewriting technique, expansion through generalization, is required to reduce the false positive results.

References

Arenas, M., Barceló, P., Fagin, R., & Libkin, L. (2004, June). Locally consistent transformations and query answering in data exchange. In Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (pp. 229-240). ACM.

Sekhavat asgharzadeh, Y. (2014). A framework for information integration using ontological foundations (Doctoral dissertation, Memorial University of Newfoundland).

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007).Dbpedia: A nucleus for a web of open data (pp. 722-735). Springer Berlin Heidelberg

Bellahsene, Z., Bonifati, A., & Rahm, E. (2011). *Schema matching and mapping* (Vol. 20): Springer.

Beneventano, D., Bergamaschi, S., Castano, S., Corni, A., Guidetti, R., Malvezzi, G., . . . Vincini, M. (2000). *Information integration: the MOMIS project demonstration.* Paper presented at the VLDB.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific american, 284*(5), 28-37.

Bunge, M. (1977). Treatise on basic philosophy: Volume 3: Ontology 1: The furniture of the world. *Reidel, Boston*.

Doan, A., Domingos, P., & Halevy, A. Y. (2001). *Reconciling schemas of disparate data sources: A machine-learning approach.* Paper presented at the ACM Sigmod Record.

Doan, A., Halevy, A., & Ives, Z. (2012). *Principles of data integration*: Elsevier.

Doan, A., & Halevy, A. Y. (2005). Semantic integration research in the database community: A brief survey. *AI Magazine, 26*(1), 83.

Do, H. H., & Rahm, E. (2007). Matching large schemas: Approaches and evaluation*. Information Systems, 32(6), 857-885.*

Friedman, M., Levy, A. Y., & Millstein, T. D. (1999). Navigational plans for data integration. *AAAI/IAAI, 1999*, 67-73.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., . . . Widom, J. (1997). The TSIMMIS approach to mediation: Data models and languages. *Journal of intelligent information systems, 8*(2), 117-132.

George, D. (2005). Understanding structural and semantic heterogeneity in the context of database schema integration. *Journal of the Department of Computing, UCLAN, 4*, 29-44.

Halevy, A., Rajaraman, A., & Ordille, J. (2006). *Data integration: the teenage years.* Paper presented at the Proceedings of the 32nd international conference on Very large data bases.

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

Hernández, M. A., Miller, R. J., & Haas, L. M. (2001). Clio: A semi-automatic tool for schema mapping. *ACM Sigmod Record, 30*(2), 607.

Kirk, T., Levy, A. Y., Sagiv, Y., & Srivastava, D. (1995). *The information manifold.* Paper presented at the Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments.

Kirk, T., Levy, A. Y., Sagiv, Y., & Srivastava, D. (1995). The information manifold. Paper presented at the Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments.

Levy, A., Rajaraman, A., & Ordille, J. (1996). Querying heterogeneous information sources using source descriptions.

Lenzerini, M. (2002). Data integration: A theoretical perspective. Paper presented at the Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.

Levy, A., Rajaraman, A., & Ordille, J. (1996). Querying heterogeneous information sources using source descriptions.

Levy, A. Y. (2000). Logic-based techniques in data integration Logic-based artificial intelligence (pp. 575-595): Springer

Manolescu, I., Florescu, D., & Kossmann, D. (2001). *Answering XML Queries on Heterogeneous Data Sources.* Paper presented at the VLDB.

Parsons, J., & Wand, Y. (2000). Emancipating instances from the tyranny of classes in information modeling. *ACM Transactions on Database Systems (TODS), 25*(2), 228-268.

Parsons, J., & Wand, Y. (2003). Attribute-based semantic reconciliation of multiple data sources *Journal on Data Semantics I* (pp. 21-47): Springer.

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal, 10*(4), 334-350.

San, R. (2012). *Ventana research 2012 value index for data integration.* (Research). CA, USA: Ventana Research.

Wang, T., & Pottinger, R. (2008). SeMap*: a generic mapping construction system.* Paper presented at the Proceedings of the 11th international conference on Extending database technology: Advances in database technology.