University of London
Imperial College of Science, Technology and Medicine
Department of Computing

Schema Matching and Schema Merging based on Uncertain Semantic Mappings

Nikos Rizopoulos

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the University of London and
the Diploma of Imperial College, November 2009

Abstract

This dissertation lies in the research area of **schema integration**: the problem of combining the data of different data sources by creating a **unified** representation of these data. Two core issues in schema integration are **schema matching**, *i.e.* the identification of correspondences, or **mappings**, between input schema objects, and **schema merging**, *i.e.* the creation of a unified schema based on the identified mappings. Examples of mappings found in the literature include **semantic mappings**, *e.g.* "author represents the same concept as writer", and **data mappings**, *e.g.* "each data value of name is equal to the concatenation of a first-name value and a last-name value". In this dissertation, we propose a schema integration framework which (1) is only concerned with semantic mappings (that associate schema objects based on simple set based comparisons of the objects' instances) and which (2) explicitly represents and manages the **uncertainty** as to which semantic relationship is the correct one to use in any mapping. In our framework, we adopt a wide set of semantic mappings that allow for a precise, rigorous and formal schema merging process. Our merging process produces a sound and complete integrated schema for each pair of input schemas, and in addition it generates **view definitions** between the input schemas and the integrated schema.

Acknowledgements

First and foremost, I would like to thank my supervisor, Peter McBrien, without whom I would have never completed this PhD. I would like to thank him for all his help, support and understanding he has shown throughout my PhD studies on both research and personal matters.

I would also like to thank all the colleagues and friends that helped me directly or indirectly in my research. In particular, I would like to thank Matteo Magnani, with whom I collaborated on parts presented in this dissertation. In particular, he introduced to me Dempster-Shafer's theory, we collaborated on identifying how this theory can be used to specify uncertainty on semantic mappings, and we co-wrote two papers [93, 68]. I would also like to thank George Tzallas, who was always available to listen to questions and problems, Aris Papadopoulos and Angelos Tsoukalas, for the *parties at 7 pm*, and Xenia Kleniati, for her kindness in offering me a place to stay when I was homeless.

Last but not least, I would like to thank my wife Pela for being there in both the happy and the difficult moments during my studies.

to Crene

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation lies in the research area of **schema integration**: the problem of combining the data of different data sources by creating a **unified** representation of these data. Two core issues in schema integration are **schema matching**, *i.e.* the identification of correspondences, or **mappings**, between input schema objects, and **schema merging**, *i.e.* the creation of a unified schema based on the identified mappings. Examples of mappings found in the literature include **semantic mappings** and **data mappings**. Data mappings relate the actual data values of schema objects using type conversion, string manipulation, arithmetic operations, *etc.* For example, in the relational model, a data mapping can be used to relate the data values of two date of birth attributes that use different datatypes to store data, *e.g.* one attribute could hold the date of birth using strings, such as '1 January 2010', and the other could use the DATE datatype, such as '2010-01-01'. Data mappings can also be used to relate data between several schema objects. For example, a data mapping can specify that the data values of attribute name can be derived as the concatenation of the data values of attributes first-name and last-name. Semantic mappings relate sets of data values of two schema objects using simple set comparison operators (equivalence, subsumption, intersection, *etc*) and ignoring the differences in the representation of the data values. For example, the two date of birth attributes with sets

of data values {‘1 January 2010’,‘2 January 2010’} and {‘2010-01-01’,‘2010-01-02’} are semantically equivalent.

In this dissertation, we propose a schema integration framework which (1) is only concerned with semantic mappings and which (2) explicitly represents and manages the **uncertainty** as to which is the correct semantic relationship to use in any mappings. In our framework, we adopt a wide set of semantic mappings that allow for a precise, rigorous and formal schema merging process. Our merging process produces an integrated schema for each pair of input schemas that is both sound, *i.e.* no extra data is added, and complete, *i.e.* no data is lost. In addition, our merging process generates **view definitions** between the input schemas and the integrated schema.

This chapter, in Section 1.1, describes the problems that schema integration looks at and gives an overview of our schema integration process. Section 1.2 motivates our work by outlining the complications that emerge during schema integration and the limitations of existing approaches. The contributions of our work are summarized in Section 1.3, and Section 1.4 presents the structure of the remainder of this dissertation.

1.1 The Schema Integration Task

In this section, we first define schema integration and explain the outcome of a schema integration task. Then we give an overview of the problems faced when attempting to deal with a schema integration task and how these problems can be resolved. Finally, we present an overview of the schema integration approach that we propose in this dissertation.

1.1.1 Definitions

The term that has been and will be repeatedly used throughout this dissertation is the term **schema**. By schema we mean any kind of structured or semi-structured representation of data. For example, Figure 1.1(a) illustrates schemas S_1^{er} and S_2^{er}. Schema S_1^{er} represents data about academic paper publications and schema S_2^{er} represents data about academic text book publications. Now, **schema integration** is the activity of creating a single, unified representation of the schemas of multiple data sources [60], so that these data sources can be accessed transparently. The ultimate goal of schema integration is to provide interoperability between the data sources and make the retrieval of information and knowledge more efficient. The result of schema integration is an **integrated schema** and **view definitions** between the input schemas and the integrated schema.

An example of an integrated schema is illustrated in Figure 1.1(b). The figure illustrates schema S_{3a}^{er} which is one possible integration of schemas S_1^{er} and S_2^{er} produced using previous approaches such as [18]. In S_{3a}^{er}, compatible schema objects have collapsed into a single object, *e.g.* the objects paper and book have collapsed into publication. Note that throughout this dissertation whenever we use the term **object** unqualified, we mean **schema object**. Later on we are going to see that for the same integration task our schema integration approach produces a higher quality schema S_{3b}^{er}.

View definitions specify the data mappings between the input schema objects and the integrated schema objects. In general, using the definition in [60], a data mapping is of the form $so \rightsquigarrow q_{S_x}$ and states that a query on schema object so can be partially answered using query q_{S_x} on schema S_x, *i.e.* that the result of query q_{S_x} is a subset of the instances of so. Depending on whether schema S_x is an input schema or the integrated schema, there are several view definition approaches [60]. For example, the view definition 1.1 below uses the **Global-As-View** (GAV) [61]

approach, where integrated schema objects are defined as views of the input schema objects. View definition 1.1 states that a query on publication can be answered partially by querying the input schema objects paper and book.

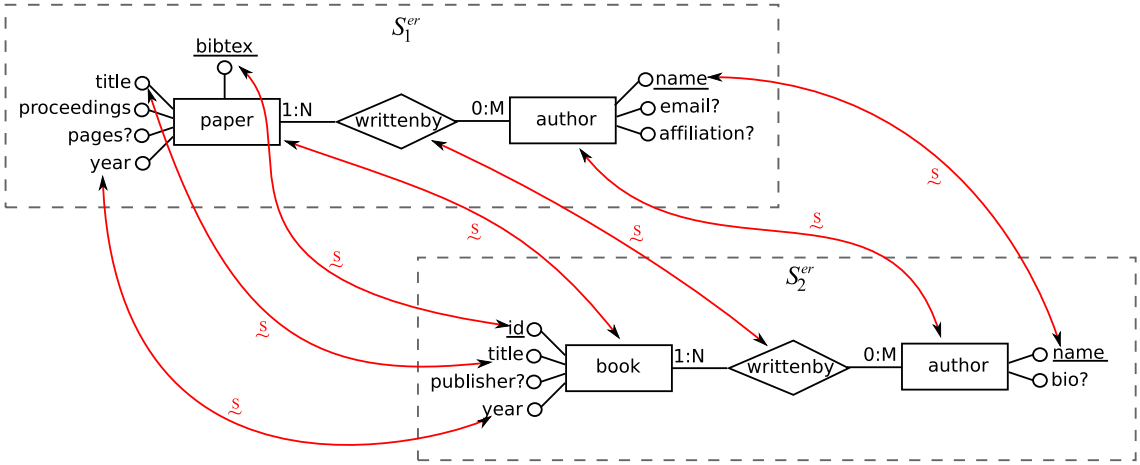
$$\begin{aligned} \langle\langle \mathsf{publication} \rangle\rangle &\rightsquigarrow \langle\langle \mathsf{paper} \rangle\rangle \\ \langle\langle \mathsf{publication} \rangle\rangle &\rightsquigarrow \langle\langle \mathsf{book} \rangle\rangle \end{aligned} \tag{1.1}$$

Later on we are going to see that the view integration approach that we have adopted in our work allows for explicitly stating whether the exact instances of a schema object, which is involved in the integration process, can be derived or not. This is important in order to produce an integrated schema that is both sound and complete, *i.e.* a schema that represents exactly the same data instances as the input schemas.

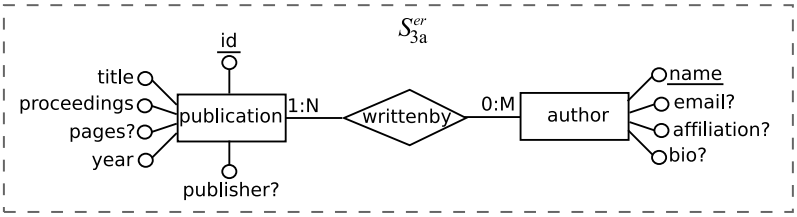
1.1.2 The problems faced within a schema integration task

One of the most challenging problems in schema integration is **heterogeneous** data sources [8]. Heterogeneity arises when dissimilarities, or conflicts, exist at any level of abstraction: the hardware used to store the data, the software used to manage them, the language used to access them, the schema used to present them (schema-level heterogeneity), and the values used to store them (data-level heterogeneity). In our research we concentrate only in schema-level heterogeneity and in particular in **semantic heterogeneity**.

Schema-level heterogeneity can arise due to the different **data model**s used to define schemas. Over the years, several different data models have been introduced for the definition of schemas. The models most frequently used are the relational model [27], the Entity-Relationship (ER) model [24], and XML [17]. Most commercial database management systems allow for an object-relational data model for data storage [105].

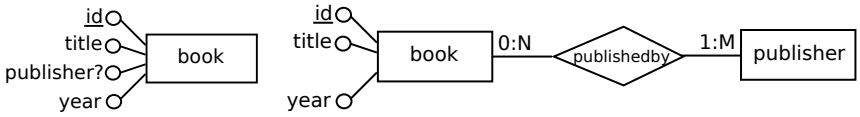


(a) Matching schemas S_1^{er} and S_2^{er}



(b) Integrated schema S_{3a}^{er}

Figure 1.1: Schema Matching and Schema Merging of two bibliography schemas for academic paper and academic text book publications



(a) Concept of publisher as an ER attribute

(b) Concept of publisher as an ER entity

Figure 1.2: Example of structural heterogeneity

Schemas defined in all of these data models can take part in a schema integration task, *e.g.* in integrating legacy databases, integrating web-services or integrating the peer schemas in a P2P DBMS [48]. Therefore, one of the issues that must be tackled by schema integration is this **model heterogeneity** problem, *e.g.* when integrating a relational schema and an XML schema.

Another type of schema-level heterogeneity which hinders the schema integration process is **structural heterogeneity** caused by the richness of the existing data models. Two schemas of the same data model can use objects of different constructs to represent the same data, thus making the identification of mappings between these objects more difficult. A very common example of structural richness of ER schemas is the use of an attribute construct instead of an entity construct to represent a set of data. This case of structural heterogeneity is illustrated in Figure 1.2. In Figure 1.2(a), the concept of a book's publisher is represented as an attribute of ER entity book, while in Figure 1.2(b) the concept of publisher is represented as an ER entity associated via a ER relationship to book. Both schemas represent identical data, but the structure of the schemas differs, which makes the integration of the two schemas harder. In our schema integration approach we are not currently concerned with structural heterogeneity *per se*; instead we focus on resolving the semantics of schema objects.

Semantic heterogeneity is a major obstacle in schema integration. By semantic heterogeneity we denote the numerous possible ways of describing the same part of

the real-world, which, when designing schemas, results in different concepts used, but not to necessarily different structures. The main reason for semantic heterogeneity is the different perspectives that designers adopt to conceptually represent a part of the real-world [8]; different concepts can be introduced depending on the level of granularity chosen or required by the application context, and several naming conventions can be employed depending on the users' background. One of the most common forms that semantic heterogeneity appears in is **descriptive conflicts** [102]. Descriptive conflicts are: (a) **homonyms**, *i.e.* representing different concepts with the same name, and (b) **synonyms** representing the same concept but with different names. For example, schemas S_1^{er} and S_2^{er} have essentially the same structure: in S_1^{er} paper is an ER entity associated via an ER relationship to ER entity author, and in S_2^{er} book is an ER entity associated via an ER relationship to ER entity author. However, the two schemas show semantic heterogeneity, because they do not represent the same data: S_1^{er} is used to represent data about academic papers, while S_2^{er} is used to represent data about academic text books.

1.1.3 Resolving the problems

To resolve schema-level heterogeneity and perform schema integration, first mappings between the input schema objects need to be identified [8]. This process is called **schema matching**. Notice that since in this work we concentrate on semantic heterogeneity we are therefore only interested in **semantic mappings** between input schema objects. Semantic mappings specify correspondences between the schema objects based on a comparison of the meaning of the concepts the schema objects represent, *e.g.* "author represents the same concept as writer", and are 1:1 mappings. We do not deal with **data mappings**, such as the 1:2 mapping "each data value of name is equal to the concatenation of a first-name value and a last-name value", nor try to resolve data scaling conflicts, data type conflicts, data precision conflicts, *etc* [55]. For example, we are not trying to identify the precise formula

that translates the values between an attribute salary, which contains instances measured in euros, and another attribute salary, which contains instances measured in pounds. The identification of a semantic mapping between these attributes can assist an expert user to define the precise data mapping. Having identified the mappings between input schema objects, the next step in schema integration is **schema merging** which produces the final integrated schema and its view definitions.

For example, consider the schemas S_1^{er} and S_2^{er} in Figure 1.1(a). Schema matching identifies the mappings between their objects. In Figure 1.1(a) the compatibility mappings are illustrated as red arrows: "author in S_1^{er} is compatible with author in S_2^{er}", which we denote as $\langle\langle\langle\langle\mathsf{author}\rangle\rangle\rangle, \overset{\mathrm{S}}{\approx}, \langle\langle\langle\mathsf{author}\rangle\rangle\rangle\rangle$, "paper is compatible with book", $\langle\langle\langle\langle\mathsf{paper}\rangle\rangle\rangle, \overset{\mathrm{S}}{\approx}, \langle\langle\langle\mathsf{book}\rangle\rangle\rangle\rangle$, "the title attribute of book is compatible with the title attribute of paper", $\langle\langle\langle\langle\mathsf{paper},\mathsf{title}\rangle\rangle\rangle, \overset{\mathrm{S}}{\approx}, \langle\langle\langle\mathsf{book},\mathsf{title}\rangle\rangle\rangle\rangle$, *etc.* Based on these mappings schema merging of S_1^{er} and S_2^{er} could be performed. An integrated schema that could be produced, S_{3a}^{er}, based on previous approaches such as [18] is illustrated in Figure 1.1(b). The integrated schema is produced by ignoring incompatibility mappings, thus all incompatible schema objects are preserved in the integrated schema, and applying a **merging rule** that states that each pair of compatible schema objects collapses into a single object in the integrated schema. In addition to the integrated schema S_{3a}^{er}, a view definition between the input schema objects and the integrated schema objects must be produced, such as the GAV definition 1.1 for the publication object shown previously.

1.1.4 Overview of our proposed approach

In our proposed schema integration approach, we have adopted the **Both-As-View** (BAV) [72] methodology to specify view definitions. BAV rules have the advantage of specifying the structure of the integrated schema, in addition to the view definitions between the input and integrated schema objects. The structure of the integrated

schema is the result of addition, deletion and rename transformations on the input schemas.

For example, regarding the mapping $\langle\langle\langle\langle\mathsf{paper}\rangle\rangle\rangle\rangle, \overset{\mathrm{S}}{\approx}, \langle\langle\langle\langle\mathsf{book}\rangle\rangle\rangle\rangle\rangle$ the merging rule in our approach could generate the following BAV definition:

$$\begin{aligned} &\mathsf{addEntity}(\langle\langle\langle\langle\mathsf{publication}\rangle\rangle\rangle\rangle, \langle\langle\langle\langle\mathsf{paper}\rangle\rangle\rangle\rangle \texttt{++} \langle\langle\langle\langle\mathsf{book}\rangle\rangle\rangle\rangle) \\ &\mathsf{contractEntity}(\langle\langle\langle\langle\mathsf{paper}\rangle\rangle\rangle\rangle, \mathit{Range}\ \texttt{Void}\ \langle\langle\langle\langle\mathsf{publication}\rangle\rangle\rangle\rangle) \\ &\mathsf{contractEntity}(\langle\langle\langle\langle\mathsf{book}\rangle\rangle\rangle\rangle, \mathit{Range}\ \texttt{Void}\ \langle\langle\langle\langle\mathsf{publication}\rangle\rangle\rangle\rangle) \end{aligned} \tag{1.2}$$

In brief, the first transformation adds an entity publication and defines its exact set of instances as the concatenation (++) of the instances of paper and book. Note that we assume set-based semantics for the instances of a schema object, thus the concatenation operator (++) we use in BAV is equivalent to a union operator. The second transformation removes the paper object. The transformation also states that the instances of the removed object cannot be derived exactly from the resulting schema. Instead, the transformation states that the instances of paper are within the range of the empty set (Void) and the instances of publication, *i.e.* that the instances of paper are a subset of the instances of publication. The third transformation similarly removes the book object. Detailed definitions of all the transformations supported in BAV are given in Section 2.3.2.

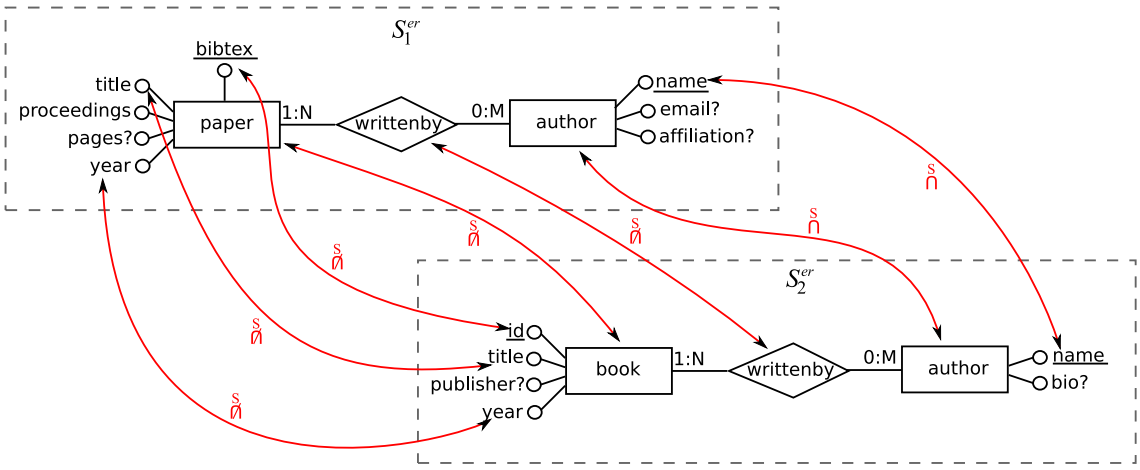
Thus, another advantage of using the BAV approach in schema integration, compared to previous approaches such as GAV, is that it allows for explicitly stating whether the exact instances of an object can be derived or not.

Taking a step back and looking at the example integrated schema S_{3a}^{er} again, we see that the integration might be criticized because it is not obvious that the input objects paper and book can be collapsed into a single object publication in the integrated schema. One could interpret that the reason the two objects have been merged into a single object is because the two objects represent the same concept.

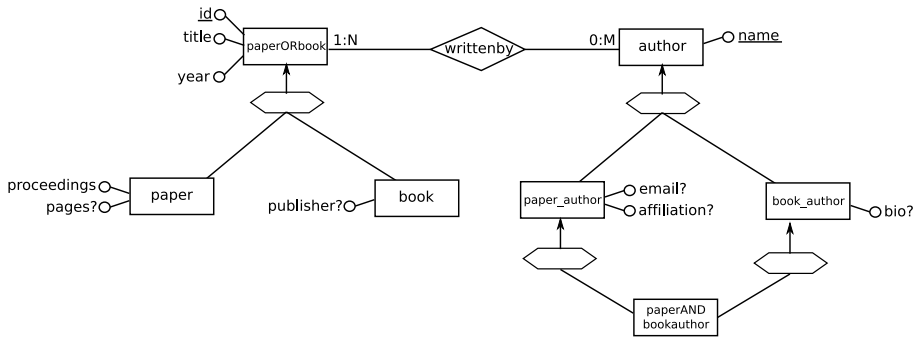
According to the mappings, however, we do not know whether the objects represent the same concept. We only know that the two objects are compatible. Compatibility mappings, in general, are imprecise, since the exact relationship between the objects is not specified. The integrated schema produced based on compatibility mappings could also be imprecise and debatable, as in this example. Therefore, it would be useful to use more precise semantic mappings to drive the schema merging process. For example, it would probably be more correct to say that the paper and book objects are disjoint, *i.e.* they represent concepts which can be generalized into a broader concept but they do not share any common instances. In the schema integration framework we propose, we require such detailed semantic mappings for the schema merging process. The advantage of using such mappings is that we can produce higher quality integrated schemas.

In particular, in our schema integration framework we specify five types of semantic mappings: equivalence ($\overset{S}{=}$), subsumption ($\overset{S}{\subset}$), intersection ($\overset{S}{\cap}$), disjointness ($\overset{S}{\not\cap}$) and incompatibility ($\overset{S}{\not\sim}$). Figure 1.3(a) illustrates such mappings between schemas S_1^{er} and S_2^{er}. Figure 1.3(b) illustrates the integrated schema S_{3b}^{er} that could be produced based on these mappings. For example the disjoint paper and book objects are unioned in a new object paperORbook. Schema S_{3b}^{er} is more precise than S_{3a}^{er} produced in our first integration attempt. In S_{3b}^{er} it is obvious which publications are papers, which ones are books, which authors have written papers, which authors have written books and which authors have written both papers and books. In our first integration in S_{3a}^{er} these concepts could not be identified.

The schema matching process in order to identify semantic mappings such the ones aforementioned, needs to identify first the **semantics** of the schema objects. By **semantics** we denote the set of real-world objects that a schema object represents. For example, we need to identify that journal-paper represents a subset of the real-world objects that paper represents. Resolving the semantics of schema objects is practically unfeasible for all possible situations, but tools that can assist in the



(a) Matching schemas S_1^{er} and S_2^{er}. Incompatibility mappings are not displayed.



(b) Integrated schema S_{3b}^{er}

Figure 1.3: Schema Matching and Schema Merging of two bibliography schemas based on more precise mappings.

process can be developed. In our research, we propose algorithms that can be applied for the identification of our five semantic mappings. The information that can be directly utilized by a tool, without any external knowledge or user intervention, is the schema structure, the names of the schema objects and the schema object instances. A tool based on this information can never be 100% certain of having identified a mapping between two schema objects correctly.

A key development in our approach is that we explicitly represent this uncertainty of the matching tool by introducing the notion of **uncertain semantic mapping**. Figure 1.4 illustrates seven uncertain mappings between the schemas S_1^{er} and S_2^{er}, *e.g.* "there is 65% certainty that paper is incompatible with book and 35% certainty that paper is disjoint with book", which we denote as $\langle \langle\!\langle \mathsf{paper} \rangle\!\rangle, [\overset{\mathrm{S}}{\not\cap}, \overset{\mathrm{S}}{\not\supset}], [.35, .65], \langle\!\langle \mathsf{book} \rangle\!\rangle \rangle$. The uncertain mapping between the writtenby relationships assigns all its certainty to disjointness, while the uncertain mappings between the author entities, and between their name attributes assign all their certainty to the equivalence mapping.

The degrees of uncertainty in each uncertain semantic mapping are derived based on:

- the available information regarding the schema objects, *e.g.* their names, their instances, the schema structure, *etc*: For example, consider the author objects. If we compare these objects based on their data instances, then the more instances we have the more certain we will be about the equivalence mapping between them.

- training data: The more training data that supports a specific semantic mapping, the more certain we are about this mapping. If there is no available training data or information regarding the schema objects, then the uncertain semantic mapping can show total ignorance, in which case our prototype system could prompt the user to select the exact semantic mapping.

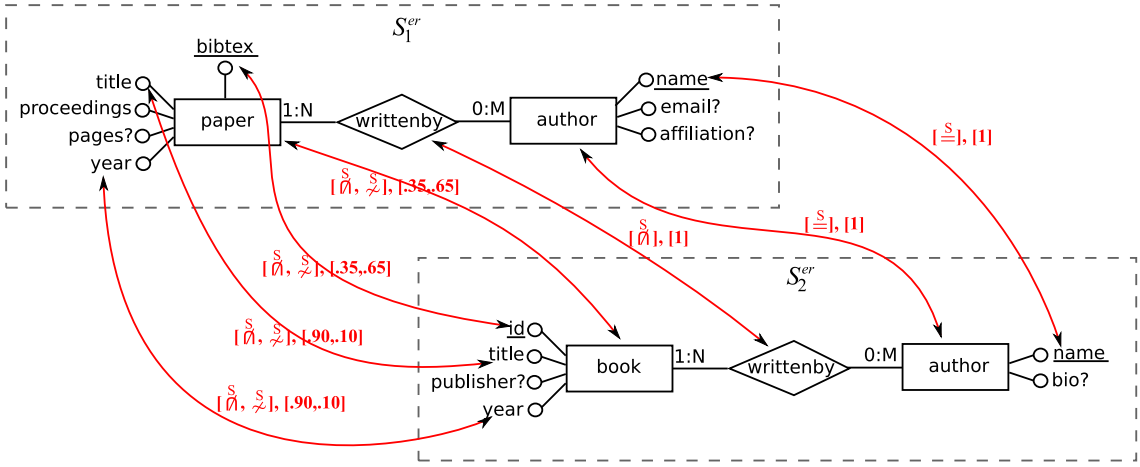


Figure 1.4: Uncertain semantic mappings between schemas S_1^{er} and S_2^{er}

Based on the uncertain semantic mappings identified during schema matching, the schema merging process can be performed. In our approach, we illustrate how schema matching uncertainty propagates into the merging process to produce a list of possible integrated schemas, instead of a single integrated schema. For example, if the only uncertain mapping between S_1^{er} and S_2^{er} is $\langle\langle$paper$\rangle\rangle$, $[\overset{S}{\emptyset}, \overset{S}{\not\supset}]$, $[.35, .65]$, $\langle\langle$book$\rangle\rangle\rangle$, the writtenby relationships are disjoint, the author entities, and the name attributes are equivalent, and all other mappings are certain incompatibility mappings then the integrated schemas produced during merging would be schemas $S_{3,0.65}^{er}$ and $S_{3,0.35}^{er}$ with 65% and 35% certainty of correctness respectively. The two integrated schemas are illustrated in Figure 1.5. $S_{3,0.65}^{er}$ corresponds to the incompatibility mapping between paper and book and $S_{3,0.35}^{er}$ corresponds to the disjointness mapping. Thus, in this particular example, the more precise schema $S_{3,0.35}^{er}$ is not the most certain one. Notice that schemas $S_{3,0.65}^{er}$, $S_{3,0.35}^{er}$ are produced based on the same merging rules used when uncertainty is not taken into consideration. These merging rules would specify the view definitions between the schemas $S_{3,0.65}^{er}$, $S_{3,0.35}^{er}$ and the input schemas S_1^{er} and S_2^{er}.

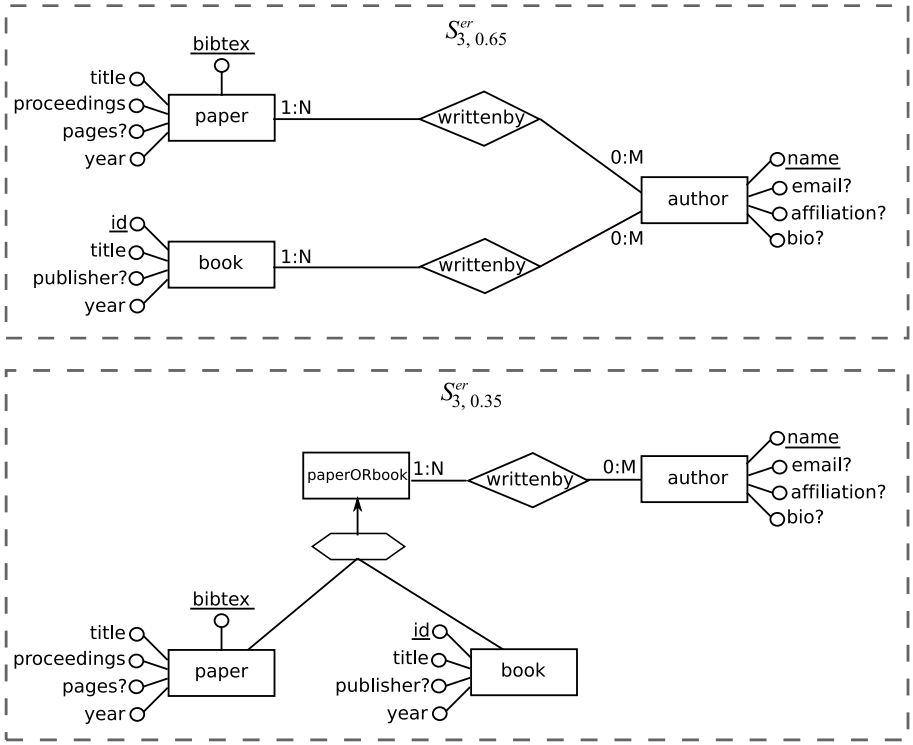


Figure 1.5: Two possible integrated schemas of S_1^{er} and S_2^{er}

1.2 Motivation

The problem of schema integration is well-known and researchers have been investigating it since the 1980's [8]. Initially, the problem arose in the context of distributed database management systems and the integration of legacy databases. These technologies provided transparent access to the data sources and allowed for a more efficient evaluation of queries. These benefits are equally important now with applications of schema integration found in every schema-based context: grid computing [103], peer-to-peer networks [48], data warehouses [82], bioinformatics [114], metasearch engines [53], digital libraries [6], geographical information systems [30], *etc.* The wide variety of applications of schema integration and the constant increase in the number of available data sources require an efficient solution to the problem.

However, schema integration, and especially the schema matching step, is a time-consuming process. When performed manually, the user has to analyze the schemas,

examine the schema object instances and specify the mapping for each pair of objects. For large schemas, this is extremely labour-intensive and in dynamic environments, where the number of sources constantly changes and the schemas evolve through time, it can be infeasible. Thus, efficient automatic or semi-automatic approaches need to be examined. This area of research is almost a decade old [90] with several assisting tools having been developed [62, 65, 73, 33, 10, 34, 109].

1.2.1 Explicitly representing uncertainty

Automatic schema matching tools discover mappings between schema objects and then automatic schema merging tools create the integrated schema. However, as we mentioned in the previous section, an automatic matching tool can never be 100% certain of having *correctly* identified the mapping between two schema objects due to the inherent uncertainty of schema matching and, thus, it is prone to errors. On the other hand, an automatic tool can significantly reduce the time spent analyzing the schemas and their schema objects and it can produce *likely* mappings. Therefore such tools can be employed to *assist* in the schema integration process.

Existing matching tools, such as the ones previously cited, propagate the uncertainty of schema matching onto the user: they discover mappings as if they are 100% certain of their correctness while the user has to be aware of the underlying uncertainty in their results; he/she needs to inspect the discovered mappings and either validate or reject them. Internally, though, a tool investigates all possible semantic mappings (equivalence, subsumption, intersection, disjointness, incompatibility) for each pair of objects and has associated levels of possibility for each mapping. For example, in Figure 1.4 the levels of possibility for seven mappings are illustrated. In existing tools, the mapping with the highest levels of possibility is the one favoured by the tool and reported to the user. Thus, the $\langle \langle\langle \mathsf{paper} \rangle\rangle, [\overset{S}{\cap}, \overset{S}{\supset}], [.35, .65], \langle\langle \mathsf{book} \rangle\rangle \rangle$ mapping in Figure 1.4 would be reported as $\langle \langle\langle \mathsf{paper} \rangle\rangle, \overset{S}{\supset}, \langle\langle \mathsf{book} \rangle\rangle \rangle$.The crucial information

that remains hidden from the user is these levels of possibility, or the levels of certainty of the tool, for each mapping. This information can help the user to either rely on the discovered mappings or investigate further some pairs of objects, and, in general, it can drive user interaction. For example, the user might want to investigate the mapping between paper and book further since the tool is not very certain of the correct mapping and realize that the correct mapping for paper and book is disjointness with 35% certainty. Instead, the tool is more certain about the mappings between the title objects and between the year objects in Figure 1.4 and the user can therefore take them as granted. Therefore, explicitly representing the uncertainty of the tool can reduce the user workload on schema matching and allow for better matching and merging results. For example, in Figure 1.5 the user can examine the uncertain integrated schema and select schema $S^{er}_{3,0.35}$ instead of $S^{er}_{3,0.65}$, even though $S^{er}_{3,0.65}$ has a higher certainty of correctness reported by the tool.

Additionally, the representation of uncertainty in schema integration can provide with more elaborate query answers, where each answer is associated with a level of certainty. Such information is essential to support keyword queries, and in environments where a schema is not always available, *e.g.* in Dataspace Support Platforms [49]. Dealing with query processing and annotation on multiple integrated schemas are out of the scope of this dissertation, but we provide a naive example next to illustrate the advantages of using our proposed schema integration framework. In [37], the authors have extensively investigated how uncertain semantic mappings can be utilized during query processing, and they have also considered the complexity issues that arise.

To illustrate their approach, let us give an example. Consider the uncertain semantic mappings $\langle\langle$publication$\rangle\rangle$, $[\overset{S}{\supset}, \overset{S}{\not\supset}]$, [.6, .4], $\langle\langle$paper$\rangle\rangle\rangle$ and $\langle\langle$publication$\rangle\rangle$, $[\overset{S}{\supset}, \overset{S}{\not\supset}]$, [.4, .6], $\langle\langle$book$\rangle\rangle\rangle$, which state that paper and book are either subsumed by or they are incompatible with publication, with the specified certainty. These two uncertain semantic mappings give rise to four cases of semantic mappings illustrated in Table 1.1, where

case	semantic mappings	certainty
#1	$\langle\langle$publication$\rangle\rangle \overset{S}{\supset} \langle\langle$paper$\rangle\rangle$	
	$\langle\langle$publication$\rangle\rangle \overset{S}{\not\supset} \langle\langle$book$\rangle\rangle$	0.36
#2	$\langle\langle$publication$\rangle\rangle \overset{S}{\not\supset} \langle\langle$paper$\rangle\rangle$	
	$\langle\langle$publication$\rangle\rangle \overset{S}{\not\supset} \langle\langle$book$\rangle\rangle$	0.24
#3	$\langle\langle$publication$\rangle\rangle \overset{S}{\supset} \langle\langle$paper$\rangle\rangle$	
	$\langle\langle$publication$\rangle\rangle \overset{S}{\supset} \langle\langle$book$\rangle\rangle$	0.24
#4	$\langle\langle$publication$\rangle\rangle \overset{S}{\not\supset} \langle\langle$paper$\rangle\rangle$	
	$\langle\langle$publication$\rangle\rangle \overset{S}{\supset} \langle\langle$book$\rangle\rangle$	0.16

Table 1.1: Four possible cases of semantic mappings

```
@inproceedings{C75,
  author={Peter P. Chen},
  title={The Entity-Relationship
         Model - Toward a Unified
         View of Data},
  year={1975},
  booktitle={VLDB},
  year={1975}
}
```

(a) An instance of a paper publication

```
@book{C99,
  author={Peter P. Chen,
          Jacky Akoka,
          Hannu Kangassalo,
          Bernhard Thalheim},
  title={Conceptual Modeling
         Current Issues and
         Future Directions},
  year={1999},
  publisher={Springer}
}
```

(b) An instance of a book

Figure 1.6: Example data

each case has a corresponding certainty. Here, the degrees of certainty are considered to be probabilities, and thus the certainty of each case in Table 1.1 is calculated as a joint probability. For example, the certainty of case #1 is calculated as the product of the certainty of $\overset{S}{\supset}$ for the pair $\langle\langle$publication$\rangle\rangle$ and $\langle\langle$paper$\rangle\rangle$ and the certainty of $\overset{S}{\not\supset}$ for the $\langle\langle$publication$\rangle\rangle$ and $\langle\langle$book$\rangle\rangle$.

Consider also the keyword query "title conference publications author Peter P. Chen" posed by a user and the two paper and book instances in Figure 1.6. Note that the query requests for titles of conference publications, *i.e. papers* in the academic jargon. However, since this is a keyword query it has to be matched to the schema available. The schema contains an object publication which has a title and an author attribute therefore the system assumes that the query requests

for the titles of publications by Peter P. Chen, and disregards the schema object paper. Query processing will now investigate all four cases of semantic mappings, and return the answer to the query. In case #1 of semantic mappings, based on the subsumption mapping between paper and publication query processing would consider only papers as publications and would associate each paper title with a 0.36 certainty of correctness, *e.g.* [{`The Entity-Relationship Model - Toward a Unified View of Data`},.36]. In case #4, only books would be considered publications and each book title would be associated with 0.16 certainty, *e.g.* [{`Conceptual Modeling Current Issues and Future Directions`},.16]. Finally, in case #3, both paper and book titles would be associated with a 0.24 certainty of correctness; #2 does not consider papers nor books as publications. The final answer to the query would be an aggregate of the above, *e.g.* [[{`The Entity-Relationship Model - Toward a Unified View of Data`},.6], [{`Conceptual Modeling Current Issues and Future Directions`},.4]] Thus, even though the query returns both paper and book titles of Peter P. Chen, papers have a higher certainty of correctness and they are presented first to the user, who actually asked for them (conference publications).

1.2.2 Using more precise semantic mappings

Apart from reducing the user workload in schema integration by explicitly representing schema matching uncertainty, another important aspect in schema integration is the quality of the integrated schemas produced. Existing merging approaches that employ only a single type of mapping, *e.g.* compatibility mappings [87] or equivalence mappings [31, 86, 75], produce integrated schemas that are imprecise and incomplete. Instead, employing a wider set of semantic mappings allows for a more precise schema integration process. For example, the integrated schema S_{3a}^{er} (Figure 1.1), which was produced based on compatibility mappings, is incomplete, since some information that was available in the input schemas S_1^{er} and S_2^{er} is lost. In particular, it is impossible in S_{3a}^{er} to identify which instances are papers and which ones

are books. Therefore, the query "give me the titles of all papers written by Peter P. Chen" on S_{3a}^{er} will be posed on object publication and will return an imprecise answer containing both paper titles, *e.g.* [{`The Entity-Relationship Model - Toward a Unified View of Data`}], and book titles, *e.g.* [{`Conceptual Modeling Current Issues and Future Directions`}]. On the other hand, when using the wider set of semantic mappings, we are able to make a more precise integration, paying attention on which objects we delete or collapse. For example, in the integrated schema S_{3b}^{er} (Figure 1.3), which was produced based on more precise semantic mappings, we are still able to make the distinction between papers and books because we have retained the original schema objects. Therefore, if the aforementioned query was posed on S_{3b}^{er} the answer would be just paper titles, *e.g.* [{`The Entity-Relationship Model - Toward a Unified View of Data`}], which is the correct answer. This precise integration potentially allows for more efficient query processing in a distributed environment, since the relationship between schema objects is more explicit and thus detecting the relevant schema objects to a query should be easier.

1.2.3 Specifying view definitions using BAV

One final aspect of schema integration is the view definitions between the input schemas and the integrated schema. Existing merging approaches [18, 89, 101] that rely on semantic mappings only generate the integrated schema. They do not generate the view definitions. Other merging approaches [87, 74], which provide view definitions, assume as input pre-defined data mappings between the schemas to be integrated. However, schema matching research that deals with the identification of such data mappings is extremely limited, due to the complexity of the problem [31].

The two most well known view definition approaches are: **Local-As-View** (LAV) and **Global-As-View** (GAV) [60]. However, instead of using these two approaches in schema integration, the more recent **Both-As-View** (BAV) approach can be

adopted, which has been shown to subsume LAV and GAV [72]. For example, in the previous section, the GAV definition 1.1 of publication can be derived from the first transformation in BAV definition 1.2. In addition, the BAV definition 1.2 can be used to derive LAV definitions of paper and book based on the contract transformations. Additionally, the use of the BAV approach gives rise to a more rigorous and formal schema merging process, since BAV definitions specify both the structure of the integrated schema and the view definitions between input and integrated schema objects. Additionally, BAV definitions explicitly state whether the instances of any schema object can be retrieved partially or fully.

1.3 Contributions of Dissertation

The two main contributions of this dissertation are:

1. The management of uncertainty in schema integration: we propose a framework based on the well-founded Dempster-Shafer's theory to explicitly represent and manage the **uncertainty** that emerges during the schema matching process regarding as to which is the correct semantic relationship between each pair of schema objects. Additionally, we explain how this schema matching uncertainty propagates into the schema merging process and how the final integrated schema is affected.

2. The definition of a low-level framework for schema merging: we propose a framework of formal, precise and rigorous low-level BAV merging rules. Based on this framework, sound and complete integrated schemas can be created. These schemas can be further improved to remove structural redundancies. The precision of our merging rules allows us to identify the cases where automating merging might be problematic. In addition, we show that the low-level framework can be extended for the integration of high-level schemas.

In more detail, our contributions are:

- we provide a definition of semantic relationships and semantic mappings based on a set-based comparison of the real-world objects schema objects represent. We also explain how semantic mappings can be translated to data mappings. Our first matching approach that attempts to identify such semantic mappings was published in [92, 15].

- we introduce the notions of uncertain semantic mapping and uncertain schema mapping. Based on these, we can explicitly represent the uncertainty of the schema matching process. Our fundamental framework behind uncertain semantic mappings was published in [93, 68].

- we show that schema matching uncertainty can be applied during the schema merging process to produce the most probable top-K schema mappings and top-K integrated schemas. We make a comparison between the top-1 and top-K approach by identifying cases where one is more preferable than the other and examining the complexity of each approach.

- we describe the implementation of our schema matching tool that supports uncertain semantic mappings and produces top-K schema mappings. We experimentally evaluate our tool and show that taking into account the schema matching uncertainty slightly improves the matching results.

- we provide a low-level schema merging framework that uses a wide set of semantic mappings and allows for a rigorous investigation of merging rules. In particular, we investigate equivalence, subsumption, intersection, disjointness and incompatibility mappings, rather that just compatibility and incompatibility mappings. The methodology behind our low-level schema merging framework was presented in [95]. The low-level framework allows for: (a) the exhaustive examination of all fundamental schema merging tasks, (b) the re-

moval of structural redundancies, and (c) reasoning about the completeness, soundness and automation of the schema merging process,

- we provide a methodology for deriving generic schema merging rules based on our low-level schema merging framework. We use the term generic throughout this dissertation to emphasize that the proposed merging process and merging rules can be applied to schemas of any data model, *i.e.* that the proposed schema merging framework is model-independent. We also provide a methodology for translating these generic merging rules into rules for high-level data models, such as the ER and the relational model. This work was published in [94].

1.4 Structure of Dissertation

This dissertation is structured as follows. Chapter 2 gives an overview of the schema integration problem and in particular of the existing schema matching and merging methodologies, including BAV. Chapter 3 explains the HDM, which is the low-level data model we have adopted. Chapter 4 introduces uncertain semantic mappings and defines the framework to manage them during matching and merging. Additionally, it explains the basic architecture of our prototype uncertain schema integration tool. Chapters 5 and 6 present our schema matching and schema merging methodologies, respectively. Chapter 5 gives the details of our schema matching implementation together with its evaluation based on experimental results. Chapter 6 introduces the low-level merging rules, their properties and their translation to high-level data models. Finally, Chapter 7 summarizes our work.

Chapter 2

Schema Integration: State of the Art

Before we begin to explain our proposed schema integration framework, it is necessary to see the current state of the art in the area and how our work relates to it. Our work concentrates on schema matching and schema merging, therefore this chapter gives a brief overview of existing research and methodologies on these topics, and compares them against our proposed approach.

The structure of this chapter is as follows. First, Section 2.1 disambiguates the different terms for schema integration that have been used in the literature and explains more precisely our definition of the terms. Then, Sections 2.2 and 2.3 discuss existing schema matching and schema merging approaches respectively, and compare them against our approach. Section 2.4 summarizes our findings.

2.1 Terminology Disambiguation

In the early 1980s, the term **view integration** [21] was used to describe the process of integrating particular types of schemas called **view**s. A **view** is a schema

that describes a subset of the data stored in a data source, from the application perspective. Thus, a single schema is used to describe the data of a data source and multiple views are defined to accommodate the types of applications that can access the data source. **View integration** focuses on the application and builds the single schema of the data source by integrating the different views.

In the mid 1980s, the advent of **distributed databases** [22] gave rise to the problem of **database integration**, *i.e.* the process of designing a global schema to represent the data of the underlying databases that constitute the distributed database. This global schema would be the result of the integration of the underlying database schemas.

The term **schema integration** was introduced in [8] as a generic term to describe both view integration and database integration. The definition of the term that we adopt is the following: schema integration is the problem of combining the data of different input schemas by (1) creating a unified representation of these data, and (2) creating view definitions between the input schemas and the unified representation.

In the late 1990s, the term **data integration** [54, 60] was introduced using a definition similar to the definition of schema integration in the previous paragraph. However, we have not adopted the term data integration because people may confuse it with the problem of identifying and merging data instances, *e.g.* tuples, strings, *etc*, that refer to the same real-world entity. This latter problem comes under different names in the literature, *e.g.* the **merge/purge problem** [52], **instance identification** [110], **duplicate elimination** [13], and is of particular importance in the data cleansing area [91, 3].

In 2003, the term **model management** [11] was introduced to describe the problem of managing generic *models*, treated as bulk objects, using high-level operators such as **Match**, **Merge**, **Diff**, *etc.* In this context, the term **model** applies to all formal descriptions and representations, *e.g.* form definitions, programming language inter-

face definitions, web site layouts, *etc*, and not just to data source schemas. In model management, the task of schema integration can be performed by first applying the Match operator to perform schema matching and identify the mappings between the schemas, and then applying the Merge operator to perform schema merging and create the integrated schema based on the result of Match.

2.2 Schema Matching

As we have already shown in the introduction, schema matching, the process of identifying mappings between schema objects, is a crucial step in schema integration. The process is extremely time-consuming if performed manually and it could even become infeasible when dealing with large evolving schemas and dynamic environments. Thus, researchers have been investigating automatic schema matching approaches.

2.2.1 Classification of Schema Matching Approaches

Until the late 1990s, little attention had been drawn on automating schema matching [90]. More recently, several approaches have been proposed and prototype tools have been developed, *e.g.* [20, 83, 62, 35, 9, 2, 38, 65, 33, 73, 10, 34, 47, 112]. In [90], a set of criteria is presented to classify schema matching approaches: (1) depending on the cardinalities of the mappings they discover, *i.e.* 1:1 mappings, 1:N mappings and M:N mappings, (2) whether they use schema-structure information or sample instances of the objects to discover mappings, (3) whether they compare schema objects based on their names, (4) if they use constraints to compare the objects, *e.g.* data type constraints, cardinalities, integrity constraints, *etc*, (5) whether they employ any auxiliary external information, *e.g.* user-input, dictionaries, ontologies, *etc*, and (6) whether a single algorithm is used to discover mappings or multiple

techniques are combined.

Examining the approaches that appear in the literature, most of them [51, 83, 35, 65, 33, 34, 73, 46]: (a) combine different types of information and techniques to compare schema objects, (b) they use the name information, (c) compare both objects and sub-structures and (d) discover 1:1 mappings. Note that some approaches claim to identify 1:N mappings by combining their discovered 1:1 mappings, *e.g.* "a is compatible with b" and "a is compatible with c", but these are not 1:N mappings from our perspective. We consider valid 1:N mappings as mappings that cannot be broken down into 1:1 mappings, *e.g.* "name equals to the concatenation of first-name and last-name".

We introduce two more criteria to classify schema matching approaches:

1. depending on whether semantic or data mappings are discovered. Semantic mappings between schema objects are derived based on the meaning, or semantics, of the concepts the schema objects represent, *e.g.* "student subsumes/is compatible with undergraduate" Data mappings are low level mappings, relating the data values of the schema objects, *e.g.* "name equals to the concatenation of first-name and last-name".

2. depending on whether schema matching uncertainty is supported for multiple semantic mappings for each pair of objects. Existing approaches either (i) discover compatibility mappings based on a similarity degree produced by comparing a specific feature of the schema objects, *e.g.* "student-ug is compatible with student-pg based on a similarity degree of 0.90 produced by comparing their names"[1], or (ii) adopt a framework that allows the representation of the tool's uncertainty on multiple semantic relationships for each pair of objects, *e.g.* "it is 30% possible that student-ug and student-pg are equivalent, 60% possible that they are either subsuming, intersecting or disjoint, and 10% possible

[1]The two strings are each made up of 10 characters and they differ only in a single position.

	compatibility	multiple semantic mappings	data mappings	supporting uncertainty on multiple semantic mappings
MUVIS [51]	✓	$\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\cap}, \overset{S}{\not\cap}$		
DIKE [83]	✓	$\overset{S}{=}, \overset{S}{\subset}$		
LSD [35]	✓			
MOMIS [9]	✓	$\overset{S}{=}, \overset{S}{\subset}, \Diamond$		
[2]	✓			
[38]	✓			
Cupid [65]	✓			
COMA [33]	✓			
SF [73]	✓			
AutoMatch [10]	✓			
GLUE [34]	✓			
[47]	✓	$\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\cap}, \overset{S}{\not\cap}$		
[112]	✓	$\overset{S}{=}, \overset{S}{\subset}$	✓	
[109]	✓			
iMAP [31]	✓		✓	
[46]	✓			fuzzy sets
[81]	✓			probabilities

Table 2.1: Classification of schema matching approaches

that they are incompatible.

In Table 2.1, we use the above two criteria to classify existing schema matching approaches. For a more elaborate classification, we also separate the approaches based on the type of semantic mappings they identify: whether they identify compatibility mappings ($\overset{S}{\sim}$), and/or more precise mappings, *i.e.* **equivalence** ($\overset{S}{=}$), **subsumption** ($\overset{S}{\subset}$), **intersection** ($\overset{S}{\cap}$), **disjointness** ($\overset{S}{\not\cap}$), **association** ($\Diamond$). Each empty cell in the table means that the corresponding feature is not supported.

The table shows that most existing approaches are concerned with identifying compatibility mappings between schema objects. Five approaches [51, 83, 9, 47, 112] discover more precise semantic mappings, two approaches deal with data mappings [112, 31] and two approaches [46, 81] adopt a framework that can support the representation of uncertainty on multiple semantic mappings for each pair of objects.

2.2.2 Types of Mappings

We separate mappings into two types: data and semantic mappings. A data mapping is practically a query over a schema. Since there is an infinite number of possible queries that can be expressed over a schema, the number of possible data mappings is also infinite. Thus, a schema matching approach that attempts to discover data mappings is practically searching in an infinite space and therefore it is trying to solve an intractable problem.

In the schema matching literature, the approach that discovers the most elaborate data mappings is iMAP [31]. There, the search space of data mappings is reduced by looking first at compatible schema objects. In particular, the first top-K most certain compatibility mappings are used. The type of data mappings discovered in iMAP include: string concatenations, *e.g.* "$\mathsf{name} = \mathit{concatenation}(\mathsf{first\text{-}name}, \mathsf{last\text{-}name})$", numerical expressions, *e.g.* "$\mathsf{list\text{-}price} = \mathsf{price} * (1 + \mathsf{tax\text{-}rate})$", unit conversions, *e.g.* "$\mathsf{weight\text{-}kg} = 2.2 * \mathsf{weight\text{-}pounds}$", schema mismatches, *e.g.* "$\mathsf{fireplace} = 1$, if $\mathsf{house\text{-}descr}$ includes the word `fireplace`", *etc.* In our proposed approach, as briefly explained in the introduction, schema matching identifies matches between schema objects with a degree of uncertainty. The matches of our approach can be used in combination with a technique such as the one presented in iMAP to reduce the search space for the possible data mappings. Thus, our proposed schema matching approach is complementary to iMAP.

Another schema matching approach that is concerned with data mappings is [112]. There, two types of data mappings are discovered: string concatenations and schema mismatches like in iMAP. The schema objects are first mapped to a domain ontology and then user-defined regular expressions are exploited to define the data mappings. Finally, in [74] data mappings are discovered as combinations of already known data mappings, thus a slightly different problem is tackled.

It is worthwhile to mention here Clio [76], which is a tool that also identifies data

mappings but only under an expert user's guidance. Therefore we do not consider Clio an automatic schema matching prototype. Note that data mappings are also covered in Section 2.3.2 from the schema merging perspective, showing how data mappings can be used to produce view definitions.

Regarding semantic mappings, they have received more attention in the schema matching literature. Several taxonomies have been proposed depending on the perspective of each approach. In [41] attention is given to schema structure with the semantic mappings having a straightforward correspondence to specific schema constructs. The mappings are **equivalence** ($\overset{S}{=}$), **subsumption** ($\overset{S}{\subset}$) and **association** ($\Diamond$), and these correspond to a node, an inclusion constraint and a link (or an ER relationship), respectively. In [64], a structural perspective, but in an object-oriented environment, is also adopted and the **aggregation** mapping is additionally proposed, *e.g.* "class is an aggregation of students". In [55], an approach is proposed that gives rise to semantic mappings between schema objects based on the context, the objects' instances, domains and roles. In addition to the previous semantic mappings, *i.e.* equivalence, subsumption, association, aggregation, objects can also be relevant if they share the same role, *e.g.* if they are both identifiers. The **incompatibility** ($\overset{S}{\not\sim}$) mapping is also explicitly defined. In [59, 102] the *real-world states* of the schema objects are compared giving rise to equivalence, subsumption, **intersection** ($\overset{S}{\cap}$) and **disjointness** ($\overset{S}{\not\cap}$) mappings. For example, "math and computing students intersect" because there are some students following a joint mathematics-computing course, while "undergraduates are disjoint with postgraduate students".

In the automatic schema matching literature, only five approaches [51, 83, 9, 47, 112] attempt to discover such elaborate semantic mappings like the ones mentioned above. Instead, most matching approaches opt to discover the more general **compatibility** ($\overset{S}{\sim}$) mapping, and leave the identification of the more precise mapping to a later stage or, more commonly, to the user.

2.2.3 Taking uncertainty into consideration

Schema matching is a well-known problem with no unique and universal solution [45]. Several research prototype tools have been proposed, but all researchers accept that the resulting matches can never be 100% correct. The main reason is that a prerequisite of schema matching is the identification of the semantics of schema objects, which is a highly intelligent process, and extremely tough and time consuming even for an expert user. Another reason that prohibits the implementation of a universal solution to the schema matching problem, is that schema matching is a highly subjective process. Even expert users who manually perform schema matching do not agree on the resulting matches. In [73] for example, the authors asked eight users to manually perform nine matching tasks and no two users could agree on the matching result for any given task. Therefore, it is highly unlikely that a matching tool can provide with matches that equally satisfy multiple users. Instead, schema matching tools are trying to provide the best matches possible and assume that the user will verify and correct the resulting matches. The user needs to be aware that the schema matching tool provides matches that may not be correct.

Even if the schema matching tool is not correct, it is interesting to know how uncertain it is for the correctness of its matches. Most existing matching approaches use algorithms that compare specific features of objects, *e.g.* their names, and produce a value in the [0..1] range for each pair. Based on these values (usually called similarity degrees) produced for all pairs, the matching approach decides which pairs are compatible. Thus, these matching approaches can at most show their uncertainty on compatibility mappings.

As far as we know, there are only two schema matching approaches [46, 81] that adopt a framework that can potentially be used to support uncertainty, not only on compatibility mappings but, on multiple more precise semantic mappings.

In [46], the approach is based on fuzzy set theory. It uses fuzzy membership func-

tions, which provide a *confidence* value in the [0..1] range for each pair of objects; the value represents the uncertainty of the tool regarding the compatibility of the two objects. Our approach compared to [46] has the additional advantage that it can be used for a wider set of semantic mappings. The requirements for each fuzzy membership function are that the function is reflexive, *i.e.* that the comparison of a schema object with itself gives the highest matching confidence, and that it is symmetrical, *i.e.* the confidence of a match for a pair of objects is irrespective of the order the two objects are compared. The confidences of multiple membership functions for each pair of objects can be combined using *max*, *min* and weighted average functions. At the moment, the approach supports fuzzy membership functions $\mu^{\mathrm{S}}_{\sim}(a_1, a_2)$, which are used to derive a value to represent the uncertainty of the tool regarding the compatibility between objects a_1 and a_2. Similarly, there could possibly be fuzzy membership functions $\mu^{\mathrm{S}}_{=}(a_1, a_2)$, $\mu^{\mathrm{S}}_{\subset}(a_1, a_2)$, *etc*, used to represent the uncertainty for other semantic mappings as well. However, the definition of these functions is a hard task, and in addition the functions would not meet the requirements, *e.g.* reflexivity is violated for disjointness, $\mu^{\mathrm{S}}_{\sqcap}(person, person) \neq 1$ since each object is equivalent and not disjoint from itself, and symmetry is violated for subsumption, $\mu^{\mathrm{S}}_{\subset}(student, person) \neq \mu^{\mathrm{S}}_{\subset}(person, student)$, where *student* is subsumed by *person* but the inverse does not hold.

Another matching approach that could support uncertainty on multiple semantic mappings is presented in [81]. The approach uses several different matching algorithms to compare schema objects and each algorithm produces a probability for each pair of objects that specifies its uncertainty on whether the two objects are compatible or not. No other semantic mappings are identified. The probabilities of all algorithms for each pair of objects are then combined using a weighted average function. Thus, the final result of the matching process is arbitrary and depends more on the confidence assigned to each algorithm, rather than the uncertainty of the tool on the result. Even though the approach currently only distributes probabil-

ity on compatibility and incompatibility mappings, potentially it could be extended to distribute probability on multiple semantic mappings.

Schema matching approaches that expose the uncertainty of the tool can produce a ranking of the schema mappings of a matching task from the most certain to the least certain schema mapping. For efficiency purposes, just the top-K schema mappings need to identified, *i.e.* the K schema mappings with the highest certainty. In [36], theoretical work about the combination of such top-K rankings of schema mappings is presented. There is no discussion about how such top-K rankings can be derived though. In [37], the authors investigate the query complexity and algorithms for answering queries when uncertain mappings are present. Additionally, an algorithm for efficiently computing the top-K answers to queries is presented. Both approaches [36, 37] are complementary to our research.

2.3 Schema Merging

During a schema integration task, the structure of the participating schemas has to be altered, or the schemas are **transformed**, to produce the final integrated schema. For example, in Figure 1.1 schemas S_1^{er} and S_2^{er} are transformed to produce the integrated schema S_{3a}^{er}: the author object of S_1^{er} is transformed to include an extra attribute bio in S_{3a}^{er}, paper and book are merged together and then renamed into publication, *etc.* However, an integrated schema is not complete until the view definitions between the integrated and the input schemas are explicitly specified, *e.g.* the view definition 1.1 in S_{3a}^{er} for object publication.

In this section, we first review schema transformation approaches and then discuss view definition approaches. Finally, we compare schema merging methodologies with our approach.

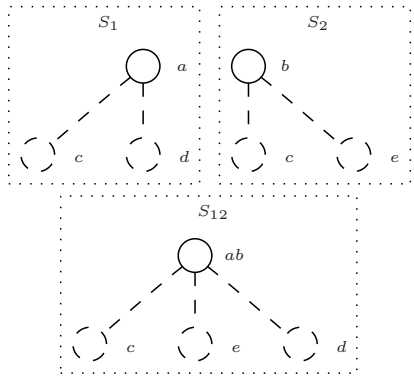
2.3.1 Schema Transformations

Several schema transformation methodologies have been proposed, especially in the schema integration field [79, 21, 7, 12, 80, 99, 59, 51, 18, 101, 86]. These approaches present some common ideas and some common transformations that we are going to illustrate next. For a more detailed comparison of the approaches, the reader can refer to [8] and [43] which are survey papers of schema transformation methodologies.

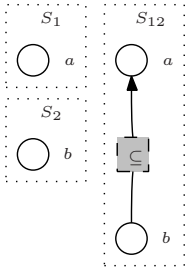
Based on [8], we separate schema transformations into two categories: (a) **conforming** and (b) **restructuring**. The goal of the conforming transformations is to make the schemas compatible, ready to be superimposed. The transformations **rename** schema objects to resolve naming conflicts and they introduce new concepts and constraints. The goal of the restructuring transformations is to improve the structure of the schema, making it more *correct*, *minimal* and *understandable* to the user [8]. Structural conflicts are resolved and redundant schema objects are deleted. In the figures below, we use full circles, which we call nodes, to represent schema objects that can appear on their own in a schema, *e.g.* entities in the ER model, relations in the relational model, *etc*, and dashed circles and lines to represent attributes. Associations or relationships between objects are represented as dark lines, which we call links, and constraints are represented as grey boxes.

Figure 2.1 illustrates the most common schema conforming transformations; objects a and b in schemas S_1 and S_2 are integrated in schema S_{12}:

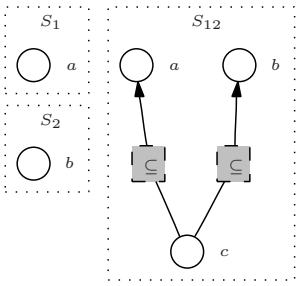
- **merging objects**: Figure 2.1(a) illustrates the most frequently used schema transformation, where the objects a and b collapse into a single object ab and their common children are also merged, *e.g.* c. This schema transformation is used in [79, 12, 80, 99, 59, 51, 18, 101, 86], where objects that represent equivalent sets of real-world objects are *combined*, *merged* or *integrated* (different terms are used in each methodology) into one object, and in [7], where the equivalent objects are *renamed* to a common name. In [21], where a very



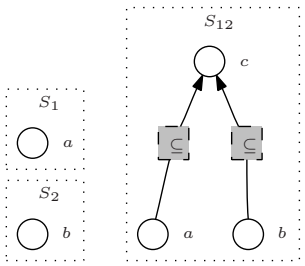
(a) $\langle\langle a \rangle\rangle \overset{S}{=} \langle\langle b \rangle\rangle$



(b) $\langle\langle a \rangle\rangle \overset{S}{\supset} \langle\langle b \rangle\rangle$



(c) $\langle\langle a \rangle\rangle \overset{S}{\cap} \langle\langle b \rangle\rangle$



(d) $\langle\langle a \rangle\rangle \overset{S}{\not\cap} \langle\langle b \rangle\rangle$

Figure 2.1: Common schema conforming transformations: schemas S_1 and S_2 are integrated into S_{12}

simple modelling language is used whereby objects can only be relations which do not have any links between them, this case is resolved by *deleting* one of the equivalent objects. Notice that in [79] this schema transformation is also used when object a subsumes object b, *i.e.* when a represents a wider set of real-world entities than b.

- **introducing inclusion constraint**: Figure 2.1(b) illustrates another frequently used schema transformation, where an inclusion constraint is introduced to show that object a subsumes object b, *i.e.* a represents a wider set of real-world entities than b. This schema transformation is used in most methodologies [21, 7, 12, 80, 99, 59, 51, 18, 101, 86], but not in [79] as explained previously.

- **introducing intersection**: Figure 2.1(c) illustrates a schema transformation usually used when the objects a and b share a common set of real-world entities in their real-world representations. This common set of real-world objects is represented by the new object c that is introduced by the transformation, which also adds the corresponding inclusion constraints that show that a and b subsume c. This schema transformation is employed in [79, 101] and implicitly introduced but not employed in [18]. Other methodologies adopt the **introducing union** transformation, that we are going to see next, in order to deal with this situation of intersecting representations.

  In our working example in the introduction, this rule was used in S_{3b}^{er} in Figure 1.3(b) to produce the intersection paperANDbookauthor object from paper_author and book_author.

- **introducing union**: Figure 2.1(d) illustrates one more common schema conforming transformation. The objects a and b are associated together by a new object c, which represents the union of the real-world sets that a and b represent. The relationships of a, b and c are explicitly specified by the two inclusion constraints between a and c, and b and c. This schema transformation

is used in [79, 7, 99, 59, 51, 101] when the objects a and b represent disjoint sets of real-world objects. As mentioned previously, this schema transformation is also employed by some methodologies [99, 59, 51] when the objects a and b are intersecting. For intersecting schema objects, [80] introduces another schema transformation that combines the **introduction of intersection** and the **introduction of union** transformations.

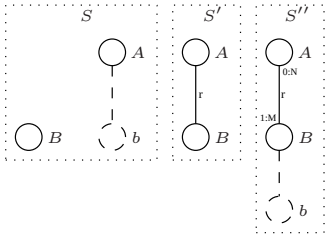
In S^{er}_{3b}, this rule was used to produce the union paperORbook object from paper and book and the union author from paper_author and book_author.

Figure 2.2 illustrates some common schema restructuring transformations; schema S is transformed to schema S':

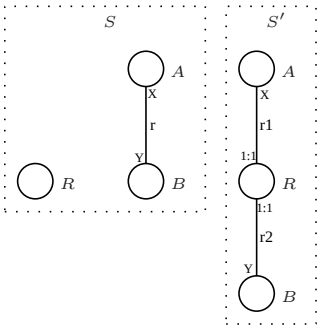
- **promoting attribute to node**: Figure 2.2(a) illustrates the most common schema restructuring transformation, where an attribute is promoted into a node. This schema transformation, employed in [101], is applied when two equivalent objects, a node and an attribute, appear in the schema, thus introducing redundancy. In the figure, node B and attribute b in S are the equivalent objects. The transformation resolves the structural conflict and removes the redundancy by collapsing the objects into one, node B in S', and introducing a new link, r.

  Similar transformations are also defined in [7], paying more attention on the type of the attribute and the cardinality constraints. In the general case, the cardinality of A in the new link r is 0:N and the cardinality of B 1:M, as shown in schema S''. In the special case, where the attribute b is a key, the cardinalities of A and B would be both 1:1.

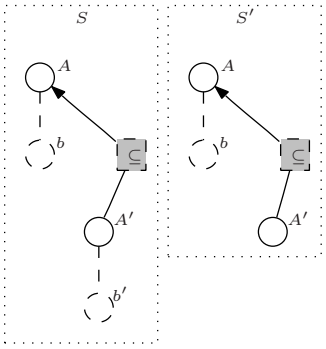
- **promoting link to node**: Figure 2.2(b) illustrates another common schema transformation, where a link object is promoted into a node and a structural conflict is resolved. Like in the previous case, the transformation is applied



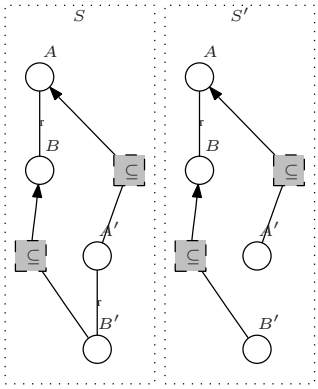
(a) $\langle\langle B\rangle\rangle \overset{\mathrm{S}}{=} \langle\langle b, A\rangle\rangle$



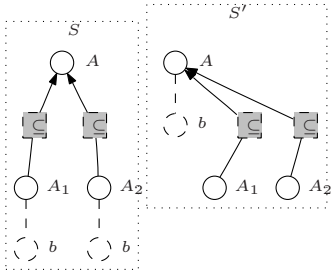
(b) $\langle\langle R\rangle\rangle \overset{\mathrm{S}}{=} \langle\langle \mathrm{r}, A, B\rangle\rangle$



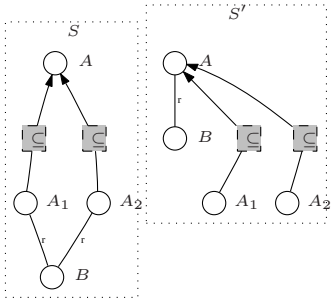
(c) $\langle\langle b, A\rangle\rangle \overset{\mathrm{S}}{\supset} \langle\langle b', A'\rangle\rangle$



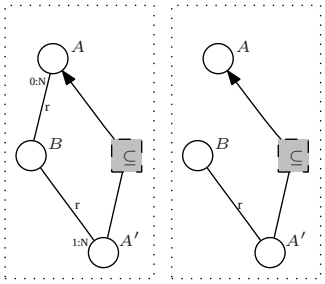
(d) $\langle\langle \mathrm{r}, A, B\rangle\rangle \overset{\mathrm{S}}{=} \langle\langle \mathrm{r}, A', B'\rangle\rangle$



(e) $\langle\langle A_1, b\rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle A_2, b\rangle\rangle$



(f) $\langle\langle \mathrm{r}, A_1, B\rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle \mathrm{r}, A_2, B\rangle\rangle$



(g) $\langle\langle \mathrm{r}, A, B\rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle \mathrm{r}, A', B\rangle\rangle$

Figure 2.2: Common schema restructuring transformations: schema S transformed to S' or S''

when two equivalent objects, a node and a link, appear in the schema. In S the equivalent objects are node R and link r and they collapse into one node R in S'. This transformation is employed in [79, 101], without considering cardinality constraints, and in [7], which is the one we illustrate here. It is also implicitly used in [80, 59].

- **removing redundant attribute**: Figure 2.2(c) illustrates a schema transformation where an attribute is deleted from the schema because it is made redundant by another attribute. In the figure, the attribute b' in S is made redundant by b and is deleted in S'. The transformation is employed in [21, 7] when an existing attribute, *e.g.* b, subsumes the redundant attribute, b'. In [59], where an exhaustive analysis is performed, the transformation is also applied when the two attribute are equivalent, intersecting or disjoint. Notice that the transformation is applicable only when the corresponding nodes A and A' are either equivalent or when A subsumes A'; the latter is the case we illustrate here, thus the inclusion constraint.

- **removing redundant link**: Figure 2.2(d) illustrates another schema transformation where a redundant link is deleted from the schema. In the figure, the link r between A' and B' in S is made redundant by link r between A and B, and is therefore deleted in S'. This transformation is implied in [59] and is used when the links are equivalent or one subsumes the other, where the subsumed link is deleted.

- **attribute generalization**: Figure 2.2(e) illustrates a schema transformation where a common attribute in multiple nodes is generalized in their common *super*-node. In the figure, the attributes b of nodes A_1 and A_2 are generalized to the attribute b of node A, which subsumes nodes A_1 and A_2, and then they are deleted. The transformation is employed in [7] when the attributes of the *sub*-nodes are disjoint, or for any type of compatible, *i.e.* equivalent, subsuming, intersecting and disjoint, attributes in [59]. In [21], the transformation is also

used without explicitly specifying the type of mapping between the attributes; we can assume that the same approach as in [59] is followed.

In our working example, in S_{3b}^{er} this rule was used to generalize attributes id, title, year and name.

- **link generalization**: Figure 2.2(f) shows a schema transformation similar to the previous one, corresponding to link objects. Here, the common link objects of the *sub*-nodes are generalized. The transformation is employed in [7, 80], for disjoint links, and in [59] for any type of compatibility mapping between the links, as long as their attached nodes (A_1 and A_2 in the figure) are generalized in a common *super*-node.

  In S_{3b}^{er}, the link generalization rule was applied for the relationship writtenby which was generalized from the entities paper_author and book_author to author.

- **removing optional link**: Figure 2.2(g) illustrates a schema transformation applied in one methodology [80]. In the figure, link **r** between nodes A and B is made redundant because of the equivalent link r between A' and B. Node A has a 0:N cardinality constraint on r, which means that not all entities represented by A participate in the link. On the other hand, node A' which is subsumed by A has a 1:N cardinality constraint with r, meaning that all entities of A' participate in r. Thus, even though the two links are equivalent the one between A and B is less specific, and thus it is deleted.

2.3.2 View Definition Approaches

In our working example, schemas S_1^{er} and S_2^{er} have been transformed into S_{3a}^{er}, which is the integrated schema that describes the underlying data sources. However, queries posed on schema S_{3a}^{er} cannot be answered if the data mappings between the schema objects of S_1^{er},S_2^{er} and S_{3a}^{er} have not been specified. Data mappings determine how queries posed to the system are answered [60]. Formally, a data mapping

is an assertion of the form:

$$so \rightsquigarrow q_S \tag{2.1}$$

where so is a schema object and q_S is a query on schema S specifying a set of instances that belong to so.

There are two main approaches for defining data mappings: (a) **local-as-view** (**LAV**) [107] and (b) **global-as-view** (**GAV**) [61]. Both of them use the term **local schema** for the schema that is tightly-coupled with a data source and the term **global schema** for the integrated schema produced when integrating all local schemas. Using this terminology in our working example, schemas S_1^{er} and S_2^{er} would be the local schemas and S_{3a}^{er} the global schema.

In LAV, the schema objects of the local schemas are described in terms of the schema objects of the global schema. This means that in the formal definition of a data mapping, Equation 2.1, so would be a local schema object and S would be the global schema. In our working example, a LAV mapping could be:

$$\langle\langle \mathsf{paper} \rangle\rangle \quad \rightsquigarrow \quad [\{x\} \mid \{x, z\} \leftarrow \langle\langle \mathsf{publication}, \mathsf{pages} \rangle\rangle; z \neq null] \tag{2.2}$$

The query on the right hand side of the above mapping states that we can generate a list of single value tuples, based on taking each binary tuple found in $\langle\langle \mathsf{publication}, \mathsf{pages} \rangle\rangle$, and filtering out those that have *null* as their second value. Note that the notation $\langle\langle \mathsf{publication}, \mathsf{pages} \rangle\rangle$ is used to represent a schema object. In this particular case, the schema object is the attribute pages of entity publication. More details about this notation can be found in Chapter 3. Hence, the above mapping specifies that we can derive instances of object paper in the local schema, from the object publication in the global schema, by identifying those publications that do not have a *null* value for the pages attribute.

The language used to define the query in the above mapping is called **IQL** [72] and it

is the query language we are going to use throughout this dissertation. IQL supports comprehensions [19], which are of the form $[e|Q_1;\ldots;Q_n]$, where e is called the *head* of the comprehension, and each Q_i is either a *filter* or a *generator*. A generator has syntax $p \leftarrow s$, where p is a pattern and s is a collection-valued expression, and returns a collection of values of pattern p. In the query above, the generator $\{x, z\} \leftarrow \langle\langle \mathsf{publication}, \mathsf{pages} \rangle\rangle$ returns a collection of values of pattern $\{x, z\}$, where x is a publication and z its pages value. A filter is a boolean-valued expression used to select the values produced by the generators that are going to be returned by the comprehension. In the query above the expression $z \neq null$ is a filter, which selects the values $\{x, z\}$ produced by the generator which have a value for z which is not *null*. Finally, a comprehension returns a collection of values of the form e. Thus, the above query, as aforementioned, will return a collection of instances x which are instances of publication, they are associated with a value z in the collection $\langle\langle \mathsf{publication}, \mathsf{pages} \rangle\rangle$, and there is the additional restriction that z is not a *null* value. Comprehensions $c_1, \ldots, c_n$ can be combined using the concatenation ++ operator, $c_1 \mathsf{++} \ldots \mathsf{++} c_n$, which returns a collection generated by concatenating the collections returned by each comprehension c_i.

In GAV the objects of the global schema are described in terms of the objects of the local schemas. This means that in Equation 2.1 *so* would be a global schema object and S would be a local schema. In our working example, two GAV mappings would be:

$$\langle\langle \mathsf{publication} \rangle\rangle \quad \rightsquigarrow \quad [\{x\} \mid \{x\} \leftarrow \langle\langle \mathsf{paper} \rangle\rangle] \tag{2.3}$$

$$\langle\langle \mathsf{publication} \rangle\rangle \quad \rightsquigarrow \quad [\{x\} \mid \{x\} \leftarrow \langle\langle \mathsf{book} \rangle\rangle] \tag{2.4}$$

The query $[\{x\} \mid \{x\} \leftarrow \langle\langle \mathsf{paper} \rangle\rangle]$ returns all x that are instances of $\langle\langle \mathsf{paper} \rangle\rangle$. In the view definition 1.1 in the introduction, we used a shorthand for this query using only the schema object $\langle\langle \mathsf{paper} \rangle\rangle$. In the rest of this thesis, whenever a schema object *so* is used on its own in a query, then this implies $[\{x_1, \ldots, x_n\} \mid \{x_1, \ldots, x_n\} \leftarrow so]$.

The above GAV mapping specifies that we can derive instances of publication from the paper and book objects.

The advantage of the LAV approach against the GAV approach is that it favors the extensibility of the system: to add a new data source to the system, the database engineer just has to specify the mappings of the objects of the new source's schema against the existing global schema; no changes to any existing mappings are necessary. In GAV an addition of a new source means that the existing mappings have to be amended. The same applies when removing a data source from the system. The advantage of GAV against LAV is in query processing, which in GAV is a simple unfolding process [60], while in LAV query processing is a well-known non-trivial problem [88].

Mappings of both LAV and GAV can be classified into three categories based on the cover of correct answers they retrieve [60]: (a) **sound**, (b) **complete** and (c) **exact**. Assume that the complete set of correct answers for a schema object *so* is q. A mapping $so \rightsquigarrow q_S$ is (a) sound, if $q_S \subseteq q$, *i.e.* a subset of the correct answers is retrieved by the mapping (b) complete, if $q \supseteq q_S$, *i.e.* the mapping retrieves all the correct answers in addition to some incorrect ones, and (c) exact, if $q = q_S$, if only correct answers are retrieved and no correct answer is missing.

In our working example, the LAV mapping 2.2 is a complete, but not an exact, mapping. This mapping selects from the object publication in S_{3a}^{er} all its instances that have a non-*null* valued attribute for pages in order to retrieve the instances of paper of S_1^{er}. Looking at the schemas S_1^{er} and S_2^{er}, pages is an attribute of paper but not an attribute of book. Thus, book instances in S_{3a}^{er} have *null* valued pages; non-*null* valued pages are only associated with paper instances. However, there might be instances of paper in S_1^{er} whose pages attribute has not been defined; these instances will also have *null* valued pages and thus will not be retrieved by the mapping 2.2. Therefore, the mapping retrieves a subset of the correct instances of paper, which based on the definitions above, asserts a sound but not an exact mapping.

In [44], a new approach, called **GLAV**, is introduced for defining data mappings. GLAV combines the expressive powers of both LAV and GAV but query processing using GLAV mappings is still as hard as in LAV. A GLAV mapping is of the form:

$$q_L \rightsquigarrow q_G$$

where q_L is a query on a local schema and q_G is a query on the global schema, *i.e.* it extends Equation 2.1 by allowing a query on the left-hand side of the mapping. GLAV was proposed in a web data context and web data require navigation on arbitrarily long paths. To deal with this issue, GLAV extends LAV and GAV by allowing q_L to be a recursive datalog query. More details on GLAV can be found in [44].

All of the above approaches face the problem that when the local or the global schemas change, or *evolve*, their existing mappings need to be amended. In [72] a new approach, called **both-as-view** (**BAV**), is presented, which subsumes both LAV and GAV and deals with schema evolution efficiently. A mapping in BAV defines both the schema transformation that should be performed on the schema and the data mapping related to the schema object that is being transformed. The BAV rule can be either a LAV or a GAV mapping and in addition it specifies whether the mapping is exact or not. Formally, a BAV mapping takes one of the following two forms:

$$\mathsf{transformationConstruct}(so, Range\ q_l\ q_u), or$$

$$\mathsf{renameConstruct}(so, so')$$

where transformation is one of the following primitives: add,delete,extend, contract, Construct is the construct type of the object to be added, deleted, *etc*, q_l is a query on the schema S the transformation is applied on that gives the lower-bound of the instances of so and q_u is a query on S that gives the upper-bound of the instances of

so. The second form of a BAV mapping is a special case where the transformation is a rename, thus no query processing is necessary, *so* is the schema object to be renamed and *so′* is the renamed object.

The add and extend BAV transformations introduce a new object into the existing schema, while delete and contract remove an object from the schema. The difference between add and extend is that the former denotes an exact mapping while the latter denotes a non-exact mapping. Thus, in an add mapping the lower-bound query q_l and the upper-bound query q_u are identical. In an extend mapping the constant `Void` can be used to indicate that no instances of *so* can be derived from S and the constant `Any` can be used to indicate that there is no knowledge about the upper limit of instances of *so*. Similarly for delete, which denotes an exact mapping, and contract, which denotes a non-exact mapping. In the rest of this dissertation, the *Range* q_l q_u expression will be abbreviated to a single query q for the add and the delete transformations, since for these transformations $q_l = q_u$, with $q = q_l = q_u$.

In our working example, two BAV mappings between objects of S_1^{er} and S_{3a}^{er} are the following.

① extendNode($\langle\langle$publication$\rangle\rangle$, *Range* $\langle\langle$paper$\rangle\rangle$ `Any`)

② contractNode($\langle\langle$paper$\rangle\rangle$, *Range* $[\{x\} \mid \{x, z\} \leftarrow \langle\langle$publication, pages$\rangle\rangle; z \neq null]$ $\langle\langle$publication$\rangle\rangle$)

The first BAV mapping ① corresponds to the GAV mapping 2.3. The BAV mapping ② corresponds to the LAV mapping 2.2, which as it was discussed earlier is a sound but not an exact mapping. This is the reason why the delete transformation cannot be used here and instead contract is used.

In the BAV approach, the mappings between the objects of two schemas create a sequence of transformations, called a transformation **pathway**. The advantage in BAV is that pathways can be concatenated together, which is very useful in schema evolution. For example, suppose that schema S_G is the global schema and that

schema S_L is a local schema and that the mappings between S_L and S_G are given by pathway T, *i.e.* applying the transformations of T on S_L results into schema S_G. Now, assume that S_G evolves into S'_G by a new pathway T_{new}. Getting the BAV mappings between the local schema S_L and the new global schema S'_G is a simple concatenation of pathways T and T_{new}, *i.e.* $T;T_{new}$. None of the existing mappings in T needs to be altered. In the case that the local schema evolves by a transformation t into schema S'_L, then the BAV mappings between the new local schema S'_L and the global schema S_G can be obtained by prefixing pathway T with the reverse transformation of t, *i.e.* $\bar{t}$. Transformation add is the reverse of delete, extend is the reverse of contract and vice versa. The final pathway between the new local schema S'_L and the global schema S_G becomes $\bar{t};T$. Thus, the BAV approach deals more efficiently with schema evolution than LAV and GAV. More about the BAV approach can be found in [72].

2.3.3 Comparing our approach

In our schema merging approach, apart from producing the integrated schema, we require the generation of view definitions between the input and the integrated schemas. We use schema transformations as previously described and we produce integrated schemas that are sound and complete, *i.e.* our merging process does not produce any information loss nor gain. Additionally, our merging process improves the integrated schema by removing structural redundancies and thus makes the schema as minimal and understandable as possible. Finally, our approach generates BAV definitions that can be used for answering queries on the integrated schema.

In previous research, existing schema merging approaches have been concentrating in creating duplicate-free integrated schemas. In [18], schema objects which contain overlapping information are collapsed into a single object. Similarly, in [86] compatible schema objects are collapsed into a single object and multiple-type conflicts are

resolved. Both approaches only consider compatibility mappings and do not deal with more precise semantic mappings, as we do in our approach. Structural redundancies are not resolved and finally and more importantly both approaches do not generate view definitions between the input and the integrated schemas. In [59, 101] similar semantic mappings to our approach are used to integrate schemas. However again it is not explained how view definitions are produced. In [89], a generic schema merging approach is presented using both semantic and data mappings, but view definitions over the integrated schema are not generated.

Work most related to ours is presented in [87, 74, 12]. In all these approaches, the input to the schema merging process is data mappings between the input schemas. In [87], where the relational data model is used, the input data mappings specify which relations are compatible and then the algorithm combines the compatible relations into a single *overlap* relation. Any input relations that cannot be derived from the overlap relations are also added into the integrated schema. This results in an integrated schema similar to the schema our approach produces, where the input schema objects are preserved in the integrated schema. The view definitions used in [87] are specified using both the LAV and GAV approaches and thus are very similar to our BAV approach. One criticism for [87], is that it defines schema merging only for relational schemas, while our approach specifies low-level merging rules which can be translated to high-level data models, including ER, relational and XML. A general criticism for all these approaches is that they depend on given data mappings, whose discovery is widely accepted to be a very hard problem [31].

In [74], input data mappings are used to produce: (a) an integrated schema, which is a duplicate-free union of the input schemas, and (b) LAV definitions on the integrated schema. However, there is the restriction that the input data mappings can be reconstructed from the LAV definitions, *i.e.* that essentially the input data mappings *are* the view definitions. The same restriction applies in [12], where the view definitions are the same as the input data mappings used for the schema integration.

Finally, note that work on dealing with view definitions can also be found in the **data exchange** [39] literature. In data exchange, the problem that is tackled is the translation of data from an input schema to a pre-existing integrated, or *target* schema. Several research papers have been published on this topic, *e.g.* [4, 40, 67, 69]. The difference in our work, which is on schema merging and schema integration, is that the integrated schema has to be generated and does not pre-exist [57].

2.4 Summary

In this chapter, we have presented existing approaches in schema matching and schema merging, and we have compared them against our proposed approach.

One limitation of existing semantic schema matching approaches is that most of them only deal with compatibility mappings, while our approach can handle a wider set of semantic mappings, such as equivalence, subsumption, intersection and disjointness. As a result, we can produce more precise integrated schemas, as illustrated in the introduction.

The main, however, limitation of existing schema matching approaches is that they do not expose the uncertainty of their matching approach on the produced semantic mappings, even though all authors agree that the result of any matching approach is highly uncertain and erroneous. As far as we know, only two approaches expose schema matching uncertainty. The first one [46] assigns its uncertainty to a complete schema mapping, *i.e.* to the set of semantic mappings for each possible pairs of objects in a matching task. This result cannot be easily used to identify specific semantic mappings which the matching tool is least certain about and therefore the result cannot be easily corrected by the user. In [81], the levels of uncertainty assigned to each mapping represent more the confidence of the user to the matching algorithm and less the actual uncertainty of the tool about its own result.

Regarding the existing schema merging approaches, most of them only produce the integrated schema without generating any view definitions. Thus, the integrated schema cannot be used to answer any queries. Schema merging approaches that produce view definitions use data mappings as input to the merging process. However, it is widely accepted that the discovery of data mappings is an extremely hard problem. One work [87], which is more related to our approach, produces both the integrated schema and view definitions, which are also similar to ours, but the process is only defined for relational schemas, while our approach can be extended to merge schemas of multiple high-level data models.

Chapter 3

HDM: the common data model

We mentioned in the introduction that one of the issues that must be tackled in a schema integration task is the heterogeneity of the data models. To this end schema integration approaches adopt a mediator framework [111], where schemas of different data models, *e.g.* relational, ER, XML, *etc*, are first translated into into a **common data model** (**CDM**) [98], resolving any model heterogeneity problems, and then they are integrated.

The common data model used in our research is the **hypergraph data model** (**HDM**) [70, 16]. One of the main contributions of this dissertation, the definition of a low-level and formal schema merging framework, is based on the HDM (see Chapter 6). The advantage of using the HDM lies in the introduction of a generic classification of high-level data model constructs and the unambiguous translation between HDM and high-level schemas.

In this chapter, we review the HDM. Section 3.1 explains our motivation in using the HDM as our CDM. In Section 3.2, the HDM is formally defined. The generic classification of high-level model constructs based on the HDM is explained in Section 3.3. Finally, in Section 3.4, we review the translation between the ER data model and the HDM.

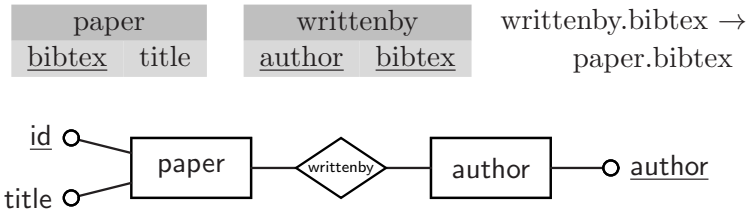
3.1 Motivation in using the HDM

When selecting a CDM, the question that arises is how expressive should the employed CDM be [96]. By expressiveness we mean the degree to which the model can directly represent a conceptualization. There are two main approaches when choosing the CDM:

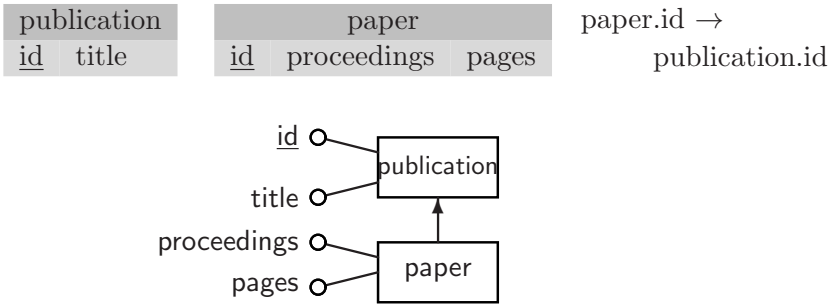
- adopting a **high-level** CDM that contains many modelling constructs, such as key, aggregation, generalization, functional dependency, *etc.* Examples of approaches that use a high-level CDM are [51, 56]. The use of multiple constructs gives more freedom when representing a conceptualization, thus making the CDM highly expressive. The disadvantage is that the translation process becomes complicated, firstly because there is rarely a simple correspondence between the constructs of high-level models (Figure 3.1) and secondly because not all constructs of a high-level model have equivalent constructs in another high-level model (Figure 3.2).

- adopting a **low-level** CDM that consists of a few simple modelling constructs, which can be used as elementary blocks to build more complex structures. Examples of approaches that use a low-level CDM are [14, 100, 26, 86]. The advantage of using a low-level CDM is that it allows for a fine grained analysis of schemas. In cases where a high-level construct does not have an equivalent construct in another high-level model, the construct can at least be partially represented using the low-level elementary constructs. The disadvantage is that schemas become larger, as they are composed of many elementary objects, and transformations between the low-level CDM schemas and high-level data model schemas become more difficult to manage.

In our schema integration approach, we have adopted a low-level CDM based on a hypergraph data structure, called the **hypergraph data model** (**HDM**) [70, 16].



(a) A relational foreign key translated to an ER relationship



(b) A relational foreign key translated to an ER subset

Figure 3.1: Problems with correspondence between high-level constructs

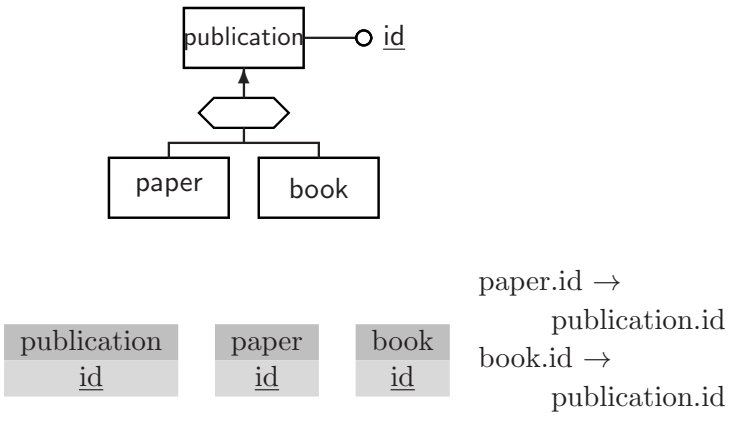


Figure 3.2: No equivalent translation of ER generalization in the relational model: the ER generalization also expresses that there are no common instances in the sub-entities, which cannot be expressed in the relational model

Using a graph structure as a CDM reduces schemas to an irreducible form [50], which in the context of relational databases has recently been identified as a sixth normal form [28, 29].

The HDM differs from other low-level CDMs in specifying a small set of fine grained constraint primitives, which can be used to build higher level constraints such as the key and cardinality constraints. Constraint primitives reduce the complexity imposed by general constraint languages, which entail the parsing, processing and comprehension of constraint expressions.

A very important feature of the HDM that we take advantage of in our schema integration framework is the generic classification of high-level model constructs that it introduces (see Section 3.3). In this generic classification, all constructs are classified into four types, which are defined based on HDM constructs. In our research, we use this generic classification to provide a methodology for deriving generic schema merging rules based on our low-level schema merging framework.

Additionally, the HDM provides an unambiguous translation of the constructs of high-level models. In [16], the translation of the relational, ER, UML and ORM constructs is defined, while in [71] the XML model is tackled. We use this work to provide a methodology for deriving specific high-level merging rules based on our generic merging rules.

3.2 Formal definition of the HDM

The HDM is a labelled, directed, nested, hypergraph structure. A **hyperedge** in the HDM is an edge that connects more than two nodes in a graph, and a hyperedge is **nested** in the sense that it can itself participate in hyperedges. Thus the definition of the HDM is recursive. In the formal definition of the HDM, we are going to use an auxiliary variable *Scheme* to make the recursion easier to understand.

Definition 3.1. *The HDM consists of three constructs: Node, Edge and Constraint. The grammar of these constructs is defined as follows:*

$$
\begin{array}{lcl}
Scheme & : & Node \mid Edge \\
Node & : & \langle\langle name \rangle\rangle \\
Edge & : & \langle\langle name, Scheme, \ldots, Scheme \rangle\rangle \\
Constraint & : & inclusion(Scheme, Scheme) \mid \\
 & & exclusion(Scheme, \ldots, Scheme) \mid \\
 & & union(Scheme, \ldots, Scheme) \mid \\
 & & mandatory(\langle Scheme, \ldots, Scheme \rangle, Scheme) \mid \\
 & & unique(\langle Scheme, \ldots, Scheme \rangle, Scheme) \mid \\
 & & reflexive(Scheme, Scheme)
\end{array}
$$

Note that the mandatory and unique constraints can be abbreviated when the tuple of schemes is unary, i.e. *mandatory(⟨Scheme⟩,Scheme) can be abbreviated to mandatory(Scheme,Scheme) and unique(⟨Scheme⟩,Scheme) can be abbreviated to unique(Scheme,Scheme).*

In the above definition of the HDM, six constraint primitives have been used, as seen in [16]. According to [16], this list of constraint primitives could be extended in the future but it is sufficient in our schema integration approach and it can deal with a wide range of data models used in practice, such as ER, relational, XML, UML and ORM. Before we explain the semantics of the six constraint primitives, we define the notion of the **HDM schema**.

Definition 3.2. *Given a set of Names that we may use for modelling the real world, an HDM schema, S, is a triple ⟨Nodes, Edges, Constraints⟩ where:*

- $Schemes = Nodes \cup Edges$

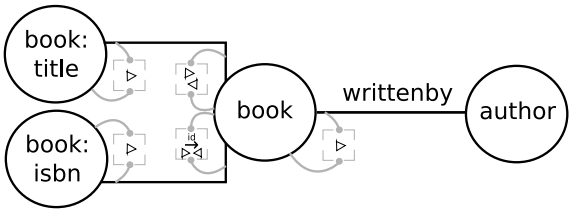


Figure 3.3: An example HDM schema

- *Nodes is a set of Node instances. Each Node instance,* $\langle\langle n_n \rangle\rangle$*, has:* $n_n \in Names$.
- *Edges is a set of Edge instances. Each Edge instance,* $\langle\langle n_e, s_1, \ldots, s_n \rangle\rangle$*, has:* $n_e \in Names \cup \{\_\}$*,* $s_1 \in Schemes$*, ...,* $s_n \in Schemes$.
- *Constraints is a set of Constraint instances. For each Constraint instance, c, the following holds:* $c \in \{inclusion(s_1, s_2)$*,* $exclusion(s_1, \ldots, s_n)$*,* $union(s_1, \ldots, s_n)$*,* $mandatory(\langle s_1, \ldots, s_n \rangle, s)$*,* $unique(\langle s_1, \ldots, s_n \rangle, s)$*,* $reflexive(s_1, s_2)\}$*, where* $s_1 \in Schemes$*, ...,* $s_n \in Schemes$ *and* $s \in Schemes$.

Note that an edge can sometimes be named using the character '_' representing an unnamed edge. An example of an HDM schema follows.

Example 3.1. Consider the HDM schema illustrated in Figure 3.3. An HDM node is represented using a circle and an HDM edge is represented using a dark line. Also, as we are later going to show in Definition 3.5, the symbol $\triangleright$ represents the *mandatory* constraint, $\triangleleft$ represents the *unique* constraint, and $\overset{\text{id}}{\rightarrow}$ represents the $reflexive$ constraint.

Therefore, we have that:

$$\begin{aligned} Nodes &= \{\langle\langle \mathsf{book} \rangle\rangle, \langle\langle \mathsf{book{:}title} \rangle\rangle, \langle\langle \mathsf{book{:}isbn} \rangle\rangle, \langle\langle \mathsf{author} \rangle\rangle\} \\ Edges &= \{\langle\langle \_, \langle\langle \mathsf{book} \rangle\rangle, \langle\langle \mathsf{book{:}title} \rangle\rangle \rangle\rangle, \\ &\quad \langle\langle \_, \langle\langle \mathsf{book} \rangle\rangle, \langle\langle \mathsf{book{:}isbn} \rangle\rangle \rangle\rangle, \\ &\quad \langle\langle \mathsf{writtenby}, \langle\langle \mathsf{book} \rangle\rangle, \langle\langle \mathsf{author} \rangle\rangle \rangle\rangle\} \end{aligned}$$

$$
\begin{aligned}
Constraints \;=\; \{&mandatory(\langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{writtenby}, \langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{author}\rangle\rangle\rangle\rangle),\\
&mandatory(\langle\langle \mathsf{book}\rangle\rangle, \langle\langle \_, \langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{book{:}title}\rangle\rangle\rangle\rangle),\\
&unique(\langle\langle \mathsf{book}\rangle\rangle, \langle\langle \_, \langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{book{:}title}\rangle\rangle\rangle\rangle),\\
&mandatory(\langle\langle \mathsf{book{:}title}\rangle\rangle, \langle\langle \_, \langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{book{:}title}\rangle\rangle\rangle\rangle),\\
&mandatory(\langle\langle \mathsf{book}\rangle\rangle, \langle\langle \_, \langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{book{:}isbn}\rangle\rangle\rangle\rangle),\\
&unique(\langle\langle \mathsf{book}\rangle\rangle, \langle\langle \_, \langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{book{:}isbn}\rangle\rangle\rangle\rangle),\\
&reflexive(\langle\langle \mathsf{book}\rangle\rangle, \langle\langle \_, \langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{book{:}isbn}\rangle\rangle\rangle\rangle),\\
&mandatory(\langle\langle \mathsf{book{:}isbn}\rangle\rangle, \langle\langle \_, \langle\langle \mathsf{book}\rangle\rangle, \langle\langle \mathsf{book{:}isbn}\rangle\rangle\rangle\rangle)\}
\end{aligned}
$$

In order to have a more compact listing of the HDM schema, $Nodes$ can lose their double chevron marks as long as this does not induce any ambiguity. In this example $Edges$ and $Constraints$ can be rewritten as follows.

$$
\begin{aligned}
Edges \;=\; \{&\langle\langle \_, \mathsf{book}, \mathsf{book{:}title}\rangle\rangle,\ \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn}\rangle\rangle,\\
&\langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{author}\rangle\rangle\}\\
Constraints \;=\; \{&mandatory(\mathsf{book}, \langle\langle \_, \mathsf{book}, \mathsf{author}\rangle\rangle),\\
&mandatory(\mathsf{book}, \langle\langle \_, \mathsf{book}, \mathsf{book{:}title}\rangle\rangle),\\
&unique(\mathsf{book}, \langle\langle \_, \mathsf{book}, \mathsf{book{:}title}\rangle\rangle),\\
&mandatory(\mathsf{book{:}title}, \langle\langle \_, \mathsf{book}, \mathsf{book{:}title}\rangle\rangle),\\
&mandatory(\mathsf{book}, \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn}\rangle\rangle),\\
&unique(\mathsf{book}, \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn}\rangle\rangle),\\
&reflexive(\mathsf{book}, \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn}\rangle\rangle),\\
&mandatory(\mathsf{book{:}isbn}, \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn}\rangle\rangle)\}
\end{aligned}
$$

◇

So far, we have defined the HDM and the HDM schema, but we have not yet given the semantics of the six constraint primitives of the HDM. The purpose of constraints is to restrict the values of schema objects in a schema. Thus, in order to explain the HDM constraint primitives we first need to define what the values, or **extent**s, of nodes and edges can be, and what makes an **HDM schema instance**.

Definition 3.3. *Given an HDM schema* $S = \langle Nodes, Edges, Constraints\rangle$*, an*

instance I of S is a structure for which there exists a function $Ext_{S,I} : Schemes \rightarrow P(Seq(Vals))$ where: $Vals$ is the set of values we wish to model as being in the domain of our schema, Seq gives a sequence of those values, and P forms the power set of those sequences. We also have the following restrictions:

1. *each tuple $\langle a_1, \ldots, a_m \rangle \in Ext_{S,I}(\langle\langle N_e, s_1, \ldots, s_m \rangle\rangle)$ has: $a_i \in Ext_{S,I}(s_i)$, for all $1 \leq i \leq m$*

2. *for every $c \in Constraints$, c holds.*

We call $Ext_{S,I}(s)$ the extent of $s \in Schemes$.

Note that the first restriction in Definition 3.3 enforces that the extent of an edge is drawn from values present in the extents of nodes and other edges it connects. The second restriction refers to the constraints of the schema and it implies that the extents of the objects do not violate the constraints. We will discuss in detail when a constraint holds in Definition 3.5, where the semantics of the constraint primitives are defined. Note that no restriction is put on any particular instance in the extent of an object; this would form the basis of typing in the HDM. Finally note that the semantics of the HDM schema and HDM schema instance are set based and hence at present we can only use the HDM to accurately model data models with set based semantics.

Now that we have defined the HDM schema instance, we can define the semantics of the HDM constraints. To this end we need an auxiliary **project** function that produces a view of an HDM edge restricted to contain a subset of the nodes or edges that edge connects. In the following definitions any variable s_i denotes a member of *Schemes*. Note that for each definition of a constraint primitive we give both a functional form, *e.g.* $inclusion(s_1, s_2)$, and an equivalent shorthand infix form, *e.g.* $s_1 \subseteq s_2$.

Definition 3.4. *The HDM project function* $\pi_{S,I}(\langle s_x, \ldots, s_y \rangle), s, t)$*, takes a tuple of schemes* $\langle s_x, \ldots, s_y \rangle$ *that must appear in edge* s*, together with a tuple* t *that appears in the extent of* s*,* $Ext_{S,I}(s)$*, and returns the values in tuple* t *that correspond to the schemes* $\langle s_x, \ldots, s_y \rangle$*:*

$$\pi(\langle s_x, \ldots, s_y \rangle, \langle\langle n_e, s_1, \ldots, s_x, \ldots, s_y, \ldots, s_n \rangle\rangle, \langle a_1, \ldots, a_x, \ldots, a_y, \ldots, a_n \rangle) = \langle a_x, \ldots, a_y \rangle,$$

where $\langle a_1, \ldots, a_x, \ldots, a_y, \ldots, a_n \rangle \in Ext_{S,I}(\langle\langle n_e, s_1, \ldots, s_x, \ldots, s_y, \ldots, s_n \rangle\rangle)$*.*

For a singleton tuple of schemes, i.e. $\langle s_x \rangle$*, the project function* $\pi_{S,I}(\langle s_x \rangle, s, t)$ *can be abbreviated into* $\pi_{S,I}(s_x, s, t)$*.*

Definition 3.5. *The six constraint primitives in the definition of the HDM, have the following semantics:*

1. $inclusion(s_1, s_2) \equiv s_1 \subseteq s_2 \equiv \langle\langle \subseteq, s_1, s_2 \rangle\rangle$

   States that the extent of s_1 *is always a subset of the extent of* s_2*,* i.e.

   for all I*:* $Ext_{S,I}(s_1) - Ext_{S,I}(s_2) = \emptyset$*.*

2. $exclusion(s_1, \ldots, s_n) \equiv s_1 \not\cap \ldots \not\cap s_n \equiv \langle\langle \not\cap, s_1, \ldots, s_n \rangle\rangle$

   States that the extent of $s_1, \ldots, s_n$ *are disjoint,* i.e.

   for all $1 \leq x < y \leq n$*, and for all* I*:* $Ext_{S,I}(s_x) \cap Ext_{S,I}(s_y) = \emptyset$*.*

3. $union(s, s_1, \ldots, s_n) \equiv s = s_1 \cup \ldots \cup s_n \equiv \langle\langle \cup, s, s_1, \ldots, s_n \rangle\rangle$

   States that the extent of s *can be derived by taking the union of the extents of* $s_1, \ldots, s_n$*,* i.e.

   for all I*:* $Ext_{S,I}(s) = Ext_{S,I}(s_1) \cup \ldots \cup Ext_{S,I}(s_n)$*.*

4. $mandatory(\langle s_1, \ldots, s_n \rangle, s) \equiv \langle s_1, \ldots, s_n \rangle \rhd s \equiv \langle\langle \rhd, \langle s_1, \ldots, s_n \rangle, s \rangle\rangle$

   States that each combination $\{a_1, \ldots, a_n\}$ *of the values* a_i *in the extent of* s_i*,* $1 \leq i \leq n$*, must appear at least once in the extent of the edge* s *that connects the* s_i*'s,* i.e.

   for all I*:* $\{\{a_1, \ldots, a_n\} \mid a_1 \in Ext_{S,I}(s_1) \wedge \ldots \wedge a_n \in Ext_{S,I}(s_n)\} -$

   $$\{\{\pi_{S,I}(s_1, s, t), \ldots, \pi_{S,I}(s_n, s, t\}) \mid t \in Ext_{S,I}(s)\} = \emptyset$$

5. $unique(\langle s_1, \ldots, s_n \rangle, s) \equiv \langle s_1, \ldots, s_n \rangle \lhd s \equiv \langle\langle \lhd, \langle s_1, \ldots, s_n \rangle, s \rangle\rangle$

   States that every combination of the values in the extents of $s_1, \ldots, s_n$ *must appear at most once in the extent of the edge* s *that connects them,* i.e.

   for all I: $\{t \mid t \in Ext_{S,I}(s) \wedge t' \in Ext_{S,I}(s) \wedge t \neq t' \wedge$

   $$\pi_{S,I}(s_1, s, t) = \pi_{S,I}(s_1, s, t') \wedge \ldots \wedge \pi_{S,I}(s_n, s, t) = \pi_{S,I}(s_n, s, t')\} = \emptyset$$

6. $reflexive(s_1, s_2) \equiv s_1 \overset{id}{\rightarrow} s_2 \equiv \langle\langle \overset{id}{\rightarrow}, s_1, s_2 \rangle\rangle$

   States that if an instance of scheme s_1 *appears in the extent of the edge* s_2 *then one of the instances of* s_2 *must be an identity tuple,* i.e.

   for all I: $\{\pi_{S,I}(s_1, s_2, t) \mid t \in Ext_{S,I}(s_2)\} -$

   $$\{\pi_{S,I}(s_1, s_2, t) \mid t \in Ext_{S,I}(s_2) \wedge$$
   $$t = \langle \pi_{S,I}(s_1, s_2, t), \pi_{S,I}(s_1, s_2, t) \rangle\} = \emptyset$$

Example 3.2. To illustrate these constraints, consider the HDM schema in Example 3.1 and in particular the nodes ⟨⟨book⟩⟩,⟨⟨author⟩⟩,⟨⟨book:isbn⟩⟩ and the edges
⟨⟨writtenby, book, author⟩⟩, ⟨⟨_, book, book:isbn⟩⟩. Suppose there are three data sources with this schema but different instances I_1, I_2, I_3 for which:

$\forall x, x \in \{1, 2, 3\}$:

$Ext_{S,I_x}(\langle\langle \text{book} \rangle\rangle)$ = {{`The Relational Model for Database Management`}, {`Conceptual Modeling Current Issues and Future Directions`}}

$Ext_{S,I_x}(\langle\langle \text{book:isbn} \rangle\rangle)$ = {{`0201141922`}, {`3540659269`}}

$Ext_{S,I_x}(\langle\langle \text{author} \rangle\rangle)$ = {{`E. F. Codd`}, {`P. P. Chen`}}

$Ext_{S,I_1}(\langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{author} \rangle\rangle) = \{$

$\{$`Conceptual Modeling Current Issues and Future Directions`,

`P. P. Chen`$\}\}$

$Ext_{S,I_1}(\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle) = \{$

$\{$`The Relational Model for Database Management`, `0201141922`$\}$

$\{$`Conceptual Modeling Current Issues and Future Directions`,

`3540659269`$\}\}$

$Ext_{S,I_2}(\langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{author} \rangle\rangle) = \{$

$\{$`The Relational Model for Database Management`, `E. F. Codd`$\}$,

$\{$`Conceptual Modeling Current Issues and Future Directions`,

`P. P. Chen`$\}\}$

$Ext_{S,I_2}(\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle) = \{$

$\{$`The Relational Model for Database Management`, `0201141922`$\}$

$\{$`The Relational Model for Database Management`, `3540659269`$\}\}$

$Ext_{S,I_3}(\langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{author} \rangle\rangle) = \{$

$\{$`The Relational Model for Database Management`, `E. F. Codd`$\}$,

$\{$`Conceptual Modeling Current Issues and Future Directions`,

`P. P. Chen`$\}\}$

$Ext_{S,I_3}(\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle) = \{$

$\{$`The Relational Model for Database Management`, `0201141922`$\}$

$\{$`Conceptual Modeling Current Issues and Future Directions`,

`3540659269`$\}\}$

The mandatory constraint $\langle\langle \mathsf{book} \rangle\rangle \rhd \langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{author} \rangle\rangle$ states that each instance of $\langle\langle \mathsf{book} \rangle\rangle$ must appear at least once in the extent of the edge $\langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{author} \rangle\rangle$, *i.e.* that each book has an author. Since there is a book, `The Relational Model for Database Management`, that is not

associated with any author in instance I_1, the mandatory constraint is not satisfied for instance I_1. Instance I_2 satisfies the aforementioned mandatory constraint.

Consider now the unique constraint $\langle\langle \mathsf{book} \rangle\rangle \lhd \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$. This constraint restricts each instance of $\langle\langle \mathsf{book} \rangle\rangle$ to appear not more than once, *i.e.* it is unique, in the extent of the edge $\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$. In I_2 this is not the case, since the same book appears twice in the extent of $\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$, thus I_2 does not satisfy the unique constraint. Instance I_1 does not have this problem.

Instance I_3 satisfies both aforementioned mandatory and unique constraints.

Finally, consider the reflexive constraint $\langle\langle \mathsf{book} \rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$. This constraint expresses that the instances of $\langle\langle \mathsf{book} \rangle\rangle$ can be identified by the instances of the edge $\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$, *i.e.* if an instance of $\langle\langle \mathsf{book} \rangle\rangle$ appears in an instance of $\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$, then the latter instance must be the identity tuple. Looking it the other way round, the reflexive constraint expresses that the extent of $\langle\langle \mathsf{book} \rangle\rangle$ is the same as the extent of $\langle\langle \mathsf{book{:}isbn} \rangle\rangle$ and that the extent of the edge between these two nodes $\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$ contains identity tuples, *e.g.* {0201141922, 0201141922}. In I_1 and I_3, we have that all the instances of $\langle\langle \mathsf{book} \rangle\rangle$ appear in the extent of $\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$:

$$\{\pi_{S,I_x}(\langle\langle \mathsf{book} \rangle\rangle, \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle, t) \mid t \in Ext_{S,I_x}(\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle)\} = Ext_{S,I_x}(\langle\langle \mathsf{book} \rangle\rangle)$$

but, there are no identity tuples in the extent of $\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$:

$$\{\pi_{S,I_x}(\langle\langle \mathsf{book} \rangle\rangle, \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle, t) \mid t \in Ext_{S,I_x}(\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle) \wedge t = \langle \pi_{S,I_x}(\langle\langle \mathsf{book} \rangle\rangle, \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle, t), \pi_{S,I_x}(\langle\langle \mathsf{book} \rangle\rangle, \langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle, t) \rangle\} = \emptyset.$$

Regarding I_2, even the first condition of the reflexive constraint does not hold; the extent of $\langle\langle \mathsf{book} \rangle\rangle$ is not the same as the extent of $\langle\langle \_, \mathsf{book}, \mathsf{book{:}isbn} \rangle\rangle$,

since one instance of ⟨⟨book⟩⟩, `Conceptual Modeling Current Issues and Future Directions`, is missing from ⟨⟨_, book, book:isbn⟩⟩.

Thus the reflexive constraint is not satisfied in any of the schema instances I_1, I_2, I_3.

◇

3.3 Supporting High-Level Data Models

Based on the HDM, a generic classification of constructs of high-level models is provided in [70]. This classification, an overview of which is given in this section, explains how the constructs of an arbitrary high-level model can be represented based on the three constructs of the HDM. In our schema merging framework, we use this classification to provide a methodology for deriving generic merging rules (see Section 6.2).

In [70], the constructs of any data model are classified as either **extentional constructs**, or **constraint constructs**, or both. An extentional construct represents a set of values from some domain. Each such construct must be built using the extentional constructs of the HDM, *i.e.* nodes and edges. A constraint construct places a restriction on extents of extentional constructs and its built using the constraint primitives of the HDM. In detail, the following constructs can be identified in any data model.

- **nodal**: a nodal construct is an extentional construct, defined independently of any other construct and it maps into a node in the HDM. Examples of nodal constructs are the entity construct in the ER model and the table construct in the relational model. A nodal schema object is identified by its scheme, which contains the name of the object, and it may appear in isolation in a schema. For example, an ER entity author is identified by the scheme ⟨⟨author⟩⟩. The

extent of a nodal schema object is equivalent to the extent of the node schema object it maps into.

- **link**: a link construct is an extentional construct defined based on other constructs and it maps into an edge in the HDM. An example of a link construct is the relationship in the ER model. The scheme of a link schema object specifies the name of the link and the schemes of the objects it associates. For example, an ER relationship haswritten, which associates ⟨⟨book⟩⟩ and ⟨⟨author⟩⟩, is defined by the scheme ⟨⟨haswritten, author, 0:N, book, 1:N⟩⟩. More details about the definition of ER constructs can be found in the next section. The extent of a link schema object is a subset of the cartesian product of the extents of the schema objects it associates.

- **link-nodal**: a link-nodal construct is an extentional construct defined based on one other construct and it maps into an edge, node and constraint combination in the HDM. An example of a link-nodal construct is the attribute in the ER model. The scheme of a link-nodal schema object contains the name of the object and the scheme of the schema object it is defined upon. For example, an ER attribute name on entity author is defined by the scheme ⟨⟨author, name, notnull⟩⟩. The extent of a link-nodal schema object is the extent of the edge it maps into, *i.e.* it is a subset of the cartesian product of the extent of the schema object the link-nodal is defined upon and the extent of the HDM node the link-nodal maps into.

- **constraint**: a constraint construct is defined based on other constructs and it maps into a combination of the constraint primitives in the HDM. An example of a constraint construct is the subset construct in the ER model. A constraint schema object does not have an extent, but instead it limits the extents of the schema objects it is associated with. The scheme of a constraint schema object contains the name of the constraint and the schemes of the schema objects, whose extent is limited. For example, an ER subset construct assigned the

Figure 3.4: An example ER schema

name book_isa_publication specifying that book is a subset of publication is defined by the scheme ⟨⟨book_isa_publication, publication, book⟩⟩.

Based on this construct classification, the constructs of a data model can be translated into HDM constructs.

3.4 Describing the ER Data Model in the HDM

In this section we are going to apply the above generic classification of constructs to the ER model.

Several variations of the ER model have been presented in the literature. The ER data model that we are going to consider in this section consists of the following constructs: entity, attribute, relationship, key, subset, and generalization. These constructs need to be classified based on the four aforementioned classes of constructs. An example of an ER schema is in Figure 3.4. The third column of Table 3.1 lists the schemes of the schema objects present in this ER schema.

An **entity** schema object is an extentional object that can be isolated in a schema, therefore the entity construct is classified as a nodal construct. To formally specify the translation of the entity construct into the HDM we use **HDM production rules**:

Definition 3.6. *An HDM production rule has the form:*

$\langle$*high level construct name*$\rangle\langle$*high level construct scheme*$\rangle \mapsto \langle$*HDM schemes*$\rangle$

$\langle$*condition*$\rangle_1 \Rightarrow \langle$*HDM constraint*$\rangle_1$

...

$\langle$*condition*$\rangle_n \Rightarrow \langle$*HDM constraint*$\rangle_n$

where

- $\langle$*high level construct scheme*$\rangle$ *is the structure used to represent a high level model construct of type* $\langle$*high level construct name*$\rangle$
- $\langle$*HDM schemes*$\rangle$ *is a list of schemes of the extentional HDM constructs the* $\langle$*high level construct scheme*$\rangle$ *maps into. This list might be empty, which is denoted using the* $\perp$ *symbol. The extent of the* $\langle$*high level construct scheme*$\rangle$ *is equivalent to the extent of the last scheme in this* $\langle$*HDM schemes*$\rangle$ *list.*
- $\langle$*condition*$\rangle$ *is a boolean expression over elements of the* $\langle$*high level construct scheme*$\rangle$*, which when satisfied causes* $\langle$*HDM constraint*$\rangle$ *to be added to the* $\langle$*HDM schemes*$\rangle$ *list.*

The production rules specifies the translation of a high level construct, $\langle$*high level construct scheme*$\rangle$*, into multiple HDM constructs, which are specified by the final* $\langle$*HDM schemes*$\rangle$ *list,* i.e. *after all* $\langle$*condition*$\rangle$*s have been checked.*

In the case of the ER entity construct the production rule is the following:

$\mathsf{entity}\langle\langle name\rangle\rangle \mapsto [\langle\langle name\rangle\rangle]$

An **attribute** construct is an extentional construct defined upon an entity, or even in some ER variations defined upon another attribute object. The attribute schema object specifies the cardinality with which the instances of the object it is defined upon can participate in the attribute, *i.e.* whether the attribute is optional or mandatory. Therefore, the attribute construct is classified as both an extentional and a

constraint construct. Its production rule, which uses an auxiliary generate_card function defined in Definition 3.7, is the following:

$$\begin{array}{lll} \textsf{attribute}\ \langle\langle entity,name,constraint\rangle\rangle \mapsto & [\langle\langle entity\_name{:}name\rangle\rangle, \\ & \langle\langle \_,entity,entity\_name{:}name\rangle\rangle] \end{array}$$

$$\begin{array}{lll} \textsf{true} & \Rightarrow & \textsf{generate\_card}(\{\langle\langle entity{:}name\rangle\rangle\}, \\ & & \langle\langle \_,entity,entity{:}name\rangle\rangle,1,*) \\ constraint = \textsf{notnull} & \Rightarrow & \textsf{generate\_card}(\{\langle\langle entity\rangle\rangle\}, \langle\langle \_, entity, entity{:}name\rangle\rangle, \\ & & 1,1) \\ constraint = \textsf{null} & \Rightarrow & \textsf{generate\_card}(\{\langle\langle entity\rangle\rangle\}, \langle\langle \_, entity, entity{:}name\rangle\rangle, \\ & & 0,1) \end{array}$$

Definition 3.7. *The generate_card($\{NE_1,\ldots,NE_n\}$,E,L,U) function generates cardinality constraints between a set of nodes or edges $\{NE_1,\ldots,NE_n\} \in Schemes$ and $E \in Edges$; there is a lower cardinality bound, L, which may be either 0 or 1, and an upper bound, U, which may be 1, $*$, or N.*

$$\begin{array}{lll} \textit{if } L = 1 & \mapsto & \{NE_1,\ldots,NE_n\} \rhd E \\ \textit{if } U = 1 & \mapsto & \{NE_1,\ldots,NE_n\} \lhd E \\ \textit{if } L \neq 1 \textit{ and } U \neq 1 & \mapsto & \bot \end{array}$$

Example 3.3. In this example, we illustrate how the ER entity $\langle\langle \textsf{book}\rangle\rangle$ and attribute $\langle\langle \textsf{book}, \textsf{title}, \textsf{notnull}\rangle\rangle$ of the example ER schema (Figure 3.4) are translated in the HDM.

1. Applying the node production rule to the entity $\langle\langle \textsf{book}\rangle\rangle$ produces an HDM node $\langle\langle \textsf{book}\rangle\rangle$.
2. Applying the attribute production rule to the attribute $\langle\langle \textsf{book}, \textsf{title}, \textsf{notnull}\rangle\rangle$ produces a node $\langle\langle \textsf{book} : \textsf{title}\rangle\rangle$ and an edge $\langle\langle \_, \textsf{book}, \textsf{book} : \textsf{title}\rangle\rangle$ initially.

class	construct	ER scheme	HDM translation
nodal	entity	⟨⟨author⟩⟩	⟨⟨author⟩⟩
link-nodal, constraint	attribute	⟨⟨author, name, notnull⟩⟩	⟨⟨author : name⟩⟩ ⟨⟨_, author, author : name⟩⟩ ⟨⟨author : name⟩⟩▷ ⟨⟨_, author, author : name⟩⟩ ⟨⟨author⟩⟩ ▷ ⟨⟨_, author, author : name⟩⟩ ⟨⟨author⟩⟩ ◁ ⟨⟨_, author, author : name⟩⟩
constraint	key	⟨⟨author, name⟩⟩	⟨⟨author⟩⟩ $\stackrel{id}{\rightarrow}$ ⟨⟨_, author, author : name⟩⟩
link-nodal, constraint	attribute	⟨⟨author, bio, notnull⟩⟩	⟨⟨author : bio⟩⟩ ⟨⟨_, author, author : bio⟩⟩ ⟨⟨author : bio⟩⟩▷ ⟨⟨_, author, author : bio⟩⟩ ⟨⟨author⟩⟩ ▷ ⟨⟨_, author, author : bio⟩⟩ ⟨⟨author⟩⟩ ◁ ⟨⟨_, author, author : bio⟩⟩
nodal	entity	⟨⟨book⟩⟩	⟨⟨book⟩⟩
link-nodal, constraint	attribute	⟨⟨book, isbn, notnull⟩⟩	⟨⟨book : isbn⟩⟩ ⟨⟨_, book, book : isbn⟩⟩ ⟨⟨book : isbn⟩⟩ ▷ ⟨⟨_, book, book : isbn⟩⟩ ⟨⟨book⟩⟩ ▷ ⟨⟨_, book, book : isbn⟩⟩ ⟨⟨book⟩⟩ ◁ ⟨⟨_, book, book : isbn⟩⟩
constraint	key	⟨⟨book, isbn⟩⟩	⟨⟨book⟩⟩ $\stackrel{id}{\rightarrow}$ ⟨⟨_, book, book : isbn⟩⟩
link-nodal, constraint	attribute	⟨⟨book, title, notnull⟩⟩	⟨⟨book : title⟩⟩ ⟨⟨_, book, book : title⟩⟩ ⟨⟨book : title⟩⟩ ▷ ⟨⟨_, book, book : title⟩⟩ ⟨⟨book⟩⟩ ▷ ⟨⟨_, book, book : title⟩⟩ ⟨⟨book⟩⟩ ◁ ⟨⟨_, book, book : title⟩⟩
link-nodal, constraint	attribute	⟨⟨book, year, null⟩⟩	⟨⟨book : year⟩⟩ ⟨⟨_, book, book : year⟩⟩ ⟨⟨book : year⟩⟩ ▷ ⟨⟨_, book, book : year⟩⟩ ⟨⟨book⟩⟩ ◁ ⟨⟨_, book, book : year⟩⟩
link, constraint	relationship	⟨⟨haswritten,author,0:N, book,1:N⟩⟩	⟨⟨haswritten, author, book⟩⟩ ⟨⟨book⟩⟩ ▷ ⟨⟨haswritten, author, book⟩⟩

Table 3.1: Schema objects in the ER schema and the translated HDM schema

The first condition of the rule, which is true for all attributes, will invoke the function generate_card($\{\langle\langle \text{book} : \text{title}\rangle\rangle\}$,$\langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$,1,*), which itself will generate the mandatory constraint $\langle\langle \text{book} : \text{title}\rangle\rangle \triangleright \langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$. This constraint expresses that each instance of node $\langle\langle \text{book} : \text{title}\rangle\rangle$ must appear at least once in the extent of the edge $\langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$, *i.e.* there is a $1 : N$ cardinality constraint between the node and the edge. This is the reason that the generate_card function is invoked with lower bound $L = 1$ and upper bound $U = *$.

Since the attribute specifies that it is mandatory, notnull, the second condition of the production rule will invoke the function generate_card($\{\langle\langle \text{book}\rangle\rangle\}$, $\langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$, 1, 1), which itself will generate the mandatory constraint $\langle\langle \text{book}\rangle\rangle \triangleright \langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$ and the unique constraint $\langle\langle \text{book}\rangle\rangle \triangleleft \langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$. The first constraint specifies that each instance of $\langle\langle \text{book}\rangle\rangle$ must appear at least once in the extent of $\langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$. The second constraint specifies that each instance of $\langle\langle \text{book}\rangle\rangle$ cannot appear more than once in the extent of $\langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$. Thus the two constraints express a 1:1 cardinality constraint between $\langle\langle \text{book}\rangle\rangle$ and $\langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$.

Thus, the attribute $\langle\langle \text{book}, \text{title}, \text{notnull}\rangle\rangle$ is translated to the following HDM objects: $\langle\langle \text{book} : \text{title}\rangle\rangle$, $\langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$, $\langle\langle \text{book} : \text{title}\rangle\rangle \triangleright \langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$, $\langle\langle \text{book}\rangle\rangle \triangleright \langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$ and $\langle\langle \text{book}\rangle\rangle \triangleleft \langle\langle \_, \text{book}, \text{book} : \text{title}\rangle\rangle$.

$\diamond$

A **relationship** construct is an extentional and a constraint construct. A relationship schema object describes an association between schema objects and it includes the cardinalities with which the instances of the associated schema objects can par-

ticipate in the relationship. Thus the relationship construct is classified as both a link and constraint construct. Its production rule is the following:

$$\begin{array}{lll} \mathsf{relationship} & \langle\langle name, entity_1, L1 : U_1, \ldots, entity_n, L_n : U_n\rangle\rangle \mapsto & \\ & [\langle\langle name, entity_1, \ldots, entity_n\rangle\rangle] & \\ \mathsf{true} & \Rightarrow \quad \mathsf{generate\_card}(\{entity_1\}, \langle\langle name, entity_1, \ldots, entity_n\rangle\rangle, L_1, U_1) & \\ \ldots & & \\ \mathsf{true} & \Rightarrow \quad \mathsf{generate\_card}(\{entity_n\}, \langle\langle name, entity_1, \ldots, entity_n\rangle\rangle, L_n, U_n) & \end{array}$$

Example 3.4. The example ER schema contains one ER relationship ⟨⟨haswritten, author, 0:N, book, 1:N⟩⟩. Applying the relationship production rule to produces the edge ⟨⟨haswritten, book, author⟩⟩ initially. The conditions of the rule will also invoke:

generate_card($\{\langle\langle$book$\rangle\rangle\}, \langle\langle$haswritten, author, book$\rangle\rangle, 1, N$) and

generate_card($\{\langle\langle$author$\rangle\rangle\}, \langle\langle$haswritten, author, book$\rangle\rangle, 0, N$).

The first call will generate the mandatory constraint ⟨⟨book⟩⟩$\triangleright$ ⟨⟨haswritten, author, book⟩⟩ while the second call will not generate any constraint constructs (see Definition 3.7).

Thus, the relationship ⟨⟨haswritten, author, 0 : N, book, 1 : N⟩⟩ is translated to the following HDM objects: ⟨⟨haswritten, author, book⟩⟩ and ⟨⟨book⟩⟩ $\triangleright$ ⟨⟨haswritten, author, book⟩⟩.

$\diamond$

The **key** construct is defined upon an entity and it serves to specify the set of attributes whose instances can be used to identify the instances of the entity. Thus, the key construct is classified as a constraint construct. Its production rule uses the notion of **natural join**, $\bowtie$, defined in Definition 3.8.

Definition 3.8. *A view over HDM edges may be formed by joining edges together to form a new virtual edge:*

$$\langle\langle E, A, B_1, \ldots, B_n\rangle\rangle \bowtie \langle\langle E, A, C_1, \ldots, C_m\rangle\rangle = \langle\langle E, A, B_1, \ldots, B_n, C_1, \ldots, C_m\rangle\rangle$$

The extent of the virtual edge is defined by a natural join over the extent of the two joined edges:

$$\begin{aligned} Ext_{S,I}(\langle\langle E, A, B_1, \ldots, B_n, C_1, \ldots, C_m\rangle\rangle) = [&\langle x, y_1, \ldots, y_n, z_1, \ldots, z_m\rangle \\ &\mid \langle x, y_1, \ldots, y_n\rangle \in Ext_{S,I}(\langle\langle E, A, B_1, \ldots, B_n\rangle\rangle) \wedge \\ &\langle x, z_1, \ldots, z_m\rangle \in Ext_{S,I}(\langle\langle E, A, C_1, \ldots, C_m\rangle\rangle)] \end{aligned}$$

The production rule for the ER key construct is:

$$\begin{aligned} &\mathsf{key}\langle\langle entity, name_1, \ldots, name_n\rangle\rangle \mapsto \bot \\ &\mathsf{true} \quad \Rightarrow \quad \langle\langle entity\rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, entity, entity : name_1\rangle\rangle \bowtie \ldots \bowtie \\ &\qquad\qquad \langle\langle \_, entity, entity : name_n\rangle\rangle \end{aligned}$$

Example 3.5. In the example ER schema, the key of the $\langle\langle \mathsf{book}\rangle\rangle$ entity is $\langle\langle \mathsf{book}, \mathsf{isbn}\rangle\rangle$. Applying the production rule to this key construct produces the reflexive constraint $\langle\langle \mathsf{book}\rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, \mathsf{book}, \mathsf{book} : \mathsf{isbn}\rangle\rangle$. This constraint expresses that the instances of node $\langle\langle \mathsf{book}\rangle\rangle$ are identified by the edge $\langle\langle \_, \mathsf{book}, \mathsf{book} : \mathsf{isbn}\rangle\rangle$, which has been produced by the translation of the $\langle\langle \mathsf{book}, \mathsf{isbn}, \mathsf{notnull}\rangle\rangle$ attribute. More specifically, if the extent of $\langle\langle \mathsf{book}\rangle\rangle$ in schema instance I_4 is:

$$Ext_{S,I_4}(\langle\langle \mathsf{book}\rangle\rangle) = \{\{\mathsf{0201141922}\}, \{\mathsf{3540659269}\}\}$$

then the extent of the edge $\langle\langle \_, \mathsf{book}, \mathsf{book} : \mathsf{isbn}\rangle\rangle$ must be:

$$\begin{aligned} Ext_{S,I_4}(\langle\langle \_, \mathsf{book}, \mathsf{book} : \mathsf{isbn}\rangle\rangle) \quad = \quad &\{\{\mathsf{0201141922}, \mathsf{0201141922}\}, \\ &\{\mathsf{3540659269}, \mathsf{3540659269}\}\} \end{aligned}$$

to satisfy the reflexive constraint.

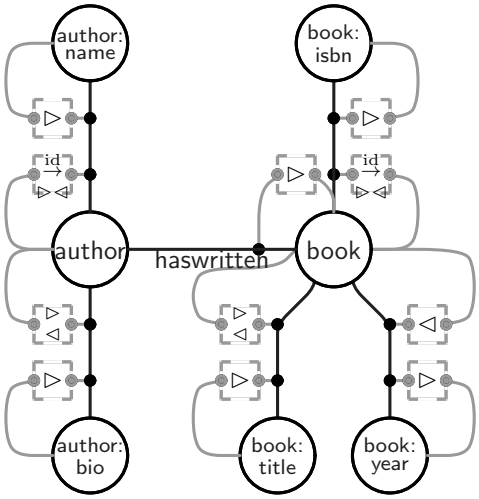


Figure 3.5: Translation of the example ER schema in the HDM

The final HDM schema produced from translating the example ER schema is illustrated in Figure 3.5. The fourth column of Table 3.1 lists the schemes of the schema object present in this HDM schema.

◇

Finally, we have the **subset** and **generalization** constructs to translate into the HDM. The subset construct specifies that the extent of an entity is a subset of the extent of another entity, *i.e.* exactly as the inclusion constraint in the HDM (Definition 3.5). Therefore, the subset construct is classified as a constraint construct. Its production rule is:

$$\mathsf{subset}\langle\langle super\_entity, sub\_entity\rangle\rangle \mapsto \bot$$

$$\mathsf{true} \quad \Rightarrow \quad \langle\langle sub\_entity\rangle\rangle \subseteq \langle\langle super\_entity\rangle\rangle$$

The generalization construct specifies that the extent of an entity is the union of the extents of other entities, which among themselves do not have any common instances. Thus, the generalization construct is classifies as a constraint construct, which has the following production rule:

$\mathsf{generalization}\langle\langle super\_entity, sub\_entity_1, \ldots, sub\_entity_n\rangle\rangle \mapsto \bot$

$\mathsf{true} \quad \Rightarrow \quad \langle\langle super\_entity\rangle\rangle = \langle\langle sub\_entity_1\rangle\rangle \cup \ldots \cup \langle\langle sub\_entity_n\rangle\rangle$

$\mathsf{true} \quad \Rightarrow \quad \langle\langle sub\_entity_1\rangle\rangle \not\cap \ldots \not\cap \langle\langle sub\_entity_n\rangle\rangle$

The first condition specifies that for each entity i, its extent is a subset of the extent of the general entity, and the last condition specifies that the n sub-entities are disjoint, *i.e.* do not have any common instances.

Example 3.6. An example of a subset is $\langle\langle \mathsf{book}, \mathsf{publication}\rangle\rangle$, which expresses that the extent of $\langle\langle \mathsf{book}\rangle\rangle$ is a subset of the extent of $\langle\langle \mathsf{publication}\rangle\rangle$. An example of a generalization is $\langle\langle \mathsf{publication}, \mathsf{book}, \mathsf{paper}\rangle\rangle$, where $\langle\langle \mathsf{publication}\rangle\rangle$ is the super-entity, whose extent subsumes the extents of the disjoint $\langle\langle \mathsf{book}\rangle\rangle$ and $\langle\langle \mathsf{paper}\rangle\rangle$.

◇

3.5 Summary

In this chapter, we have presented the hypergraph data model (HDM) which we use in our research to define our low-level schema merging framework (Chapter 6). The advantage of using the HDM is the introduction of a generic classification of high-level data model constructs, which we have also presented in this chapter. Based on this generic classification, we have shown how we can model high-level data models, such as the ER data model.

In our schema integration framework, we have used this generic classification to provide a methodology for deriving generic merging rules (Section 6.2). Additionally, we have used the modelling of high-level data models based on the generic classification to define particular high-level data model merging rules (Section 6.3).

Chapter 4

Top-K Schema Integration

In this chapter, we explain our schema integration framework.

As discussed in the introduction, one of the first steps in a schema integration task is schema matching. The purpose of schema matching is to identify correspondences between the objects of the schemas to be integrated. In our framework, the correspondences that we are interested in are **semantic mappings**. In a matching task, a list of semantic mappings for each pair of objects is called a **schema mapping**. These terms are formally defined in Section 4.2.

In our schema integration framework, we explicitly represent and manage the **uncertainty** that emerges during the schema matching process regarding the semantic mapping between each pair of schema objects. The advantages of managing uncertainty in schema integration have been explained in the literature [37, 108, 49] and our motivation in representing schema matching uncertainty has been explained in Section 1.2.1. Specifically, we require our representation of schema matching uncertainty to support the following features:

- ability to rank the possible semantic mappings for each pair of objects based on their likelihood: This is the ultimate objective for the representation of

schema matching uncertainty. By ranking the possible semantic mappings, the ones that are more likely can be investigated first and they can be used to produce integrated schemas that provide the correct query answers.

- assignment of a degree of uncertainty for each semantic mapping for each pair of objects: This is necessary for ranking semantic mappings. Additionally, these degrees of uncertainty can be used potentially to combine schema matching uncertainty with other forms of uncertainty in data sources. For example, there are several approaches [1, 23] that deal with the automated ranking of query answers and produce numerical results to represent the level of the uncertainty of each query answer, and other approaches that use numerical values to represent the uncertainty regarding the data held in a data source, *e.g.* in probabilistic databases [58]. These types of uncertainty could be potentially combined with schema matching uncertainty.

- ability to express ignorance about the possible semantic mappings for each pair of objects: Expressing ignorance is useful when we have no information about two schema objects, or when we do not want to compare the objects.

- ability to show certainty about the semantic mapping for a pair of objects.

Section 4.3 explains our approach in representing schema matching uncertainty using **uncertain semantic mapping**s (USMs), and Section 4.4 provides the rule to combine uncertain semantic mappings. Our approach, which is based on Dempster-Shafer's theory [97], is shown in Section 4.5 to support the above feature requirements and in Section 4.6, it is shown to be more suitable for the representation of schema matching uncertainty compared to other approaches used to model uncertainty.

In our schema integration framework, the final result of matching is an **uncertain schema mapping**, defined in Section 4.3. Based on an uncertain schema mapping,

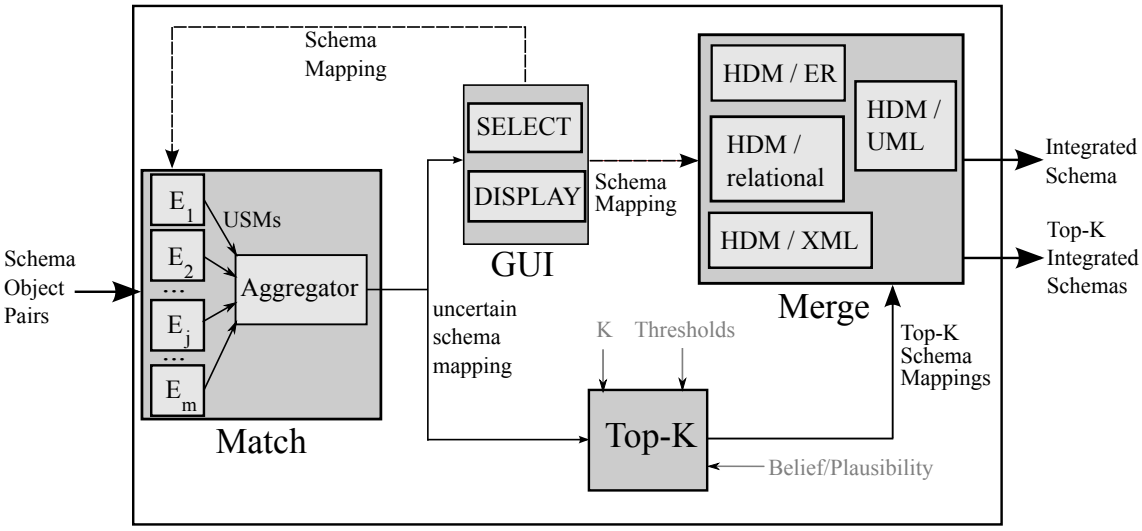


Figure 4.1: Our proposed architecture

the K most probable, *i.e.* the top-K, schema mappings can be discovered, as shown in Section 4.7. In this section, the objectives for discovering the top-1 and top-K mappings are identified and the complexity of the two processes is compared.

During schema merging, which is the next step in schema integration, the K most probable schema mappings give rise to at most K most probable integrated schemas, which we define in Section 4.8 as **top-K integrated schemas**.

Before we explain all these new terms, we describe the steps of our schema integration methodology in Section 4.1.

4.1 Top-K Integration Methodology

In this section, we describe the steps of our schema integration methodology and the architecture that implements them. The complete architecture is illustrated in Figure 4.1.

Our methodology for Top-K integration can be divided into the following steps:

1. Schema Matching: The input to our schema matching is a list of n schema

object pairs [s_i,s_j], where $s_i \in S_p$ and $s_j \in S_q$, and S_p and S_q are the schemas or sub-schemas to be integrated. The output of schema matching is an uncertain schema mapping, which is a list of uncertain semantic mappings (USMs), $[usm_1, \ldots, usm_n]$, one for each pair of objects. Our contribution to schema matching is the representation of uncertainty regarding the correct semantic relationship for each pair of schema objects [s_i,s_j] with the use of USMs. In Figure 4.1, the schema matching step is performed by the Match component, which consists of a set of experts, $E_1, \ldots, E_m$, which compare the schema objects in each pair [s_i,s_j] using different matching algorithms. Each expert e produces a list of USMs $[usm_{1,e}, \ldots, usm_{n,e}]$ which are then aggregated into the final output of schema matching, the uncertain schema mapping. In Section 4.4, based on the working example presented in the introduction (Section 1.1.4), we present Example 4.6, which describes instances of experts that might be used in the Match component, the USMs they produce and how these are combined to produce the uncertain schema mapping.

In our research, we propose a way of taking the results of each matching algorithm that each expert e uses, and derive probability masses required for the definition of USMs. Details about this derivation process can be found in Section 5.1.3. In brief, an expert user uses the GUI component to examine the results of the Match component in previous matching tasks and adjusts them to specify the correct schema mappings for these tasks. Based on these schema mappings, which we term *training data*, each expert e in the Match component identifies the probability masses for the definition of USMs.

In Section 5.3.7, we show that increasing the training set size, increases the accuracy of the USMs. However, currently, we have no other way of correlating training set size and accuracy improvement. Indeed, it is possible to propose scenarios where the training data might cause less accurate results to be produced (as mentioned in Section 5.3.4). Thus, the use of training data is

something justified empirically by the results of experimentation.

2. Schema Mapping Selection: The input for schema mapping selection is the uncertain schema mapping, $[usm_1, \ldots, usm_n]$, produced in the previous step. Based on whether we require a semi-automated integration process or a fully-automated process, the output of schema mapping selection, is either a single schema mapping or K schema mappings. Note that (as it will be explained in Section 4.7) an uncertain schema mapping specifies a set of schema mappings each associated with a probability mass. In a semi-automated process, an expert user uses the GUI component of our architecture to select the appropriate schema mapping. In our prototype implementation, no support is provided to the user, but in principle a tool, such as Clio [76], could be used by the user. In a fully-automated process, the K schema mappings with the highest likelihood values are selected. This is performed in the Top-K component, the implementation of which we describe in Section 5.2. In Section 4.7, Example 4.8 shows a case on our working example of deriving the Top-2 schema mappings.

3. Schema Merging: The input to the schema merging process is a schema mapping, supplied from the previous step, and the output is an integrated schema. For K schema mappings, at most K integrated schemas are produced. The schema merging step is performed by the Merge component of our architecture. Our contribution to schema merging is that an integrated schema and its view definitions are produced based on semantic mappings between schema objects. The process is based on merging rules on the HDM, which can be used to define rules on higher level data models, such as the ER and the relational model. More details and examples regarding schema merging are presented in Chapter 6.

Next, we define the terms uncertain semantic mapping and uncertain schema mapping. To do so, we first need to define the terms semantic mapping and schema

mapping.

4.2 Semantic Mappings

In Chapter 2, we stated that we focus on semantic schema matching, therefore the relationships that we want to discover during matching are **semantic relationships**.

4.2.1 Semantics

In order to identify the semantic relationship between two schema objects, we need to compare the semantics of the two objects. We attempt to capture the semantics of any schema object by looking at the set of real-world entities that the schema object represents. In [59], the term *real-world state* is used to represent the real-world entities of an object. We call this set of real-world entities, the **intended domain** $\mathrm{Dom}^{int}_{S,\mathrm{Inst}^{int}}(s)$ of the schema object s. In our integration framework the semantic relationship between two schema objects is defined based on the set-based comparison of the objects' intended domains.

The **intended domain** of a schema object cannot be derived from a data source, or from any other mechanical means. The data source only provides the **extent** (see Definition 3.3) of the schema object, which is an encoding of the intended domain. This encoding is frequently incorrect; it might contain errors and values may be missing or might be extraneous. The correct encoding of the intended domain of a schema object is what we call the **intended extent** of the object, which again cannot always be derived from a data source, but could be if the data source is well-maintained and known to be correct.

Definition 4.1. Intended Extent Given an HDM schema $S = \langle Nodes, Edges, Constraints\rangle$, an intended instance $\mathrm{Inst}^{int}$of S is a structure for which there exists

phd	
name	college
Dean Williams	Birkbeck
Hao Fan	Birkbeck
Lucas Zamboulis	Birkbeck
Nicos Rizopoulos	Imperial

Table 4.1: A relation of the PhD students in the AutoMed group

s	$\text{Ext}_{S_{Imp},I}(s)$
⟨⟨phd⟩⟩	[{Dean Williams}, {Hao Fan}, {Lucas Zamboulis}, {Nicos Rizopoulos}]
⟨⟨phd, name⟩⟩	[{Dean Williams,Dean Williams}, {Hao Fan,Hao Fan}, {Lucas Zamboulis,Lucas Zamboulis}, {Nicos Rizopoulos,Nicos Rizopoulos}]
⟨⟨phd, college⟩⟩	[{Dean Williams,Birkbeck}, {Hao Fan,Birkbeck}, {Lucas Zamboulis,Birkbeck}, {Nicos Rizopoulos,Imperial}]
⟨⟨phd_pk, phd, ⟨⟨phd, name⟩⟩⟩⟩	-

Table 4.2: Extents $\text{Ext}_{S_{Imp},I}$

a function $\text{Ext}^{int}_{S,\text{Inst}^{int}} : Schemes \rightarrow P(Seq(Vals))$, which provides all the correct instances intended for each $s \in Schemes$. $Vals$ is the correct set of values we wish to model as being in the domain of our schema, Seq gives a sequence of those values, and P forms the power set of those sequences. We also have the following restrictions:

1. each tuple $\langle a_1, \ldots, a_n \rangle \in \text{Ext}^{int}_{S,\text{Inst}^{int}}(\langle\langle N_e, s_1, \ldots, s_n \rangle\rangle)$ has: $a_i \in \text{Ext}^{int}_{S,\text{Inst}^{int}}(s_i)$, for all $1 \leq i \leq n$

2. for every $c \in Constraints$, c holds in Inst^{int}.

We call $\text{Ext}^{int}_{S,\text{Inst}^{int}}(s)$ the intended extent of s.

□

Example 4.1. Intended Extent Consider that the AutoMed research group has created a relational database with schema S_{Imp} storing information about its members. The database is located at Imperial College London, one of the institutions of the research group.

One of the relations in the database contains information about the PhD students of the group. Table 4.1 shows the tuples of this relation and Table 4.2 shows the schema objects of the relation (see Chapter 3) together with their extents $\text{Ext}_{S_{Imp},I}$.

In Table 4.1, the word `Nicos` is misspelled and a tuple is missing concerning new group member and PhD student Andrew Smith from Imperial College London. The extents $\text{Ext}_{S_{Imp},I}$ of the schema objects replicate these errors.

The intended extents of the objects, listed in Table 4.3, correct these errors: (a) wherever the word `Nicos` appears in the extent of an object it is replaced by the word `Nikos` in the intended extent of the object, (b) the instance {`Andrew Smith`}, which is not in $\text{Ext}_{S_{Imp},I}(\langle\langle \mathsf{phd} \rangle\rangle)$, appears in $\text{Ext}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$ and (c) the intended extents of the attributes $\langle\langle \mathsf{phd}, \mathsf{name} \rangle\rangle$ and $\langle\langle \mathsf{phd}, \mathsf{college} \rangle\rangle$ include instances associated to {`Andrew Smith`}.

◇

The intended extent $\text{Ext}^{int}_{S,\text{Inst}^{int}}(s)$ of a schema object s is a set, therefore it does not contain any duplicate elements. If s is a nodal construct, then each element of $\text{Ext}^{int}_{S,\text{Inst}^{int}}(s)$ represents a distinct real-world entity or concept. If s is a link or link-nodal construct, then each element of $\text{Ext}^{int}_{S,\text{Inst}^{int}}(s)$ represents an association of real-world entities or concepts. Between the elements of $\text{Ext}^{int}_{S,\text{Inst}^{int}}(s)$ and the

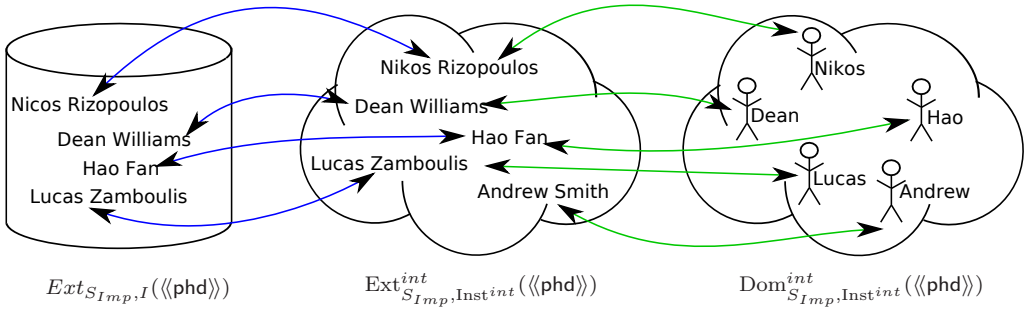
s	$\text{Ext}^{int}_{S_{Imp},\text{Inst}^{int}}(s)$
⟨⟨phd⟩⟩	[{Dean Williams}, {Hao Fan}, {Lucas Zamboulis}, {Nikos Rizopoulos}, {Andrew Smith}]
⟨⟨phd, name⟩⟩	[{Dean Williams,Dean Williams}, {Hao Fan,Hao Fan}, {Lucas Zamboulis,Lucas Zamboulis}, {Nikos Rizopoulos,Nikos Rizopoulos}, {Andrew Smith,Andrew Smith}]
⟨⟨phd, college⟩⟩	[{Dean Williams,Birkbeck}, {Hao Fan,Birkbeck}, {Lucas Zamboulis,Birkbeck}, {Nikos Rizopoulos,Imperial}, {Andrew Smith,Imperial}]
⟨⟨phd_pk, phd, ⟨⟨phd, name⟩⟩⟩⟩	-

Table 4.3: Intended Extents $\text{Ext}^{int}_{S_{Imp},\text{Inst}^{int}}$

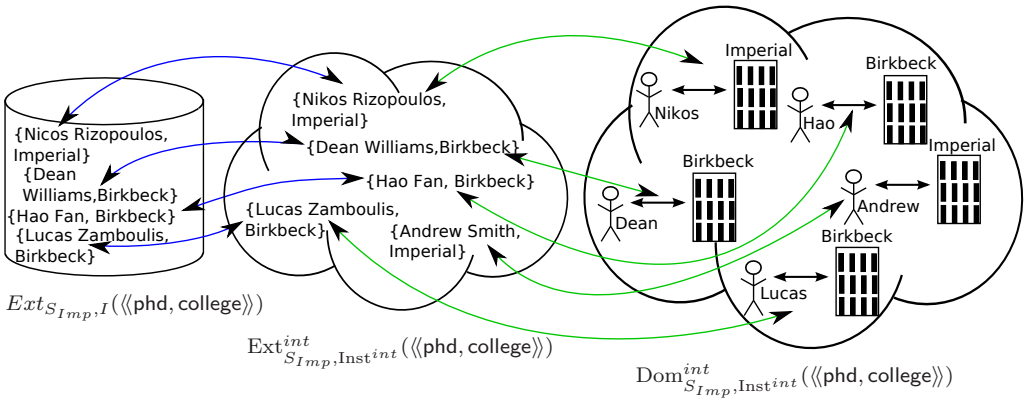
elements of $\text{Dom}^{int}_{S,\text{Inst}^{int}}(s)$, there is a 1:1 correspondence. If there is a function $map_S : \text{Ext}^{int}_{S,\text{Inst}^{int}}(s) \rightarrow \text{Dom}^{int}_{S,\text{Inst}^{int}}(s)$, that maps from the intended extent to the intended domain, then this function is bijective in schema S for object s. Note though that a real world entity might have different but correct representations in distinct schemas, *e.g.* for schemas S_1 and S_2 real world entity x can have two distinct representations y_1 and y_2 in the intended extents of the schemas, *i.e.* $map^{-1}_{S_1}(x) = y_1$, $map^{-1}_{S_2}(x) = y_2$ and $y_1 \neq y_2$, where $map^{-1}_{S_i}$ represents the inverse function of map_{S_i}. For example, a Greek citizen might have two distinct representations in a database of a Greek organization and an English organization. Both data sources could be well-maintained and correct. but in the former the Greek name of the person is used for representation, while in the latter, the name appears using Latin characters.

Definition 4.2. Intended Domain The intended domain, $\text{Dom}^{int}_{S,\text{Inst}^{int}}(s)$, of a schema object s in schema S is the set of real-world entities, concepts, or associations between entities and/or concepts, that the intended extent of s, $\text{Ext}^{int}_{S,\text{Inst}^{int}}(s)$, represents.

□



(a) Correspondences for a nodal object



(b) Correspondences for a link-nodal object

Figure 4.2: Correspondences between extents, intended extents and intended domains

Example 4.2. Intended Domain Following on from Example 4.1, the extent $\text{Ext}_{S_{Imp},I}(\langle\langle \mathsf{phd} \rangle\rangle)$ of the nodal schema object $\langle\langle \mathsf{phd} \rangle\rangle$ contains four elements in the data source.

As illustrated in Figure 4.2(a), $\text{Ext}_{S_{Imp},I}(\langle\langle \mathsf{phd} \rangle\rangle)$ corresponds to four of the elements of the intended extent $\text{Ext}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$. $\text{Ext}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$ also contains the element {`Andrew Smith`}.

Each element of $\text{Ext}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$ corresponds to a single element of $\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$ and vice versa. $\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$ contains five real-world entities, in particular five PhD students.

The intended domain $\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd}, \mathsf{college} \rangle\rangle)$ of the link-nodal schema object $\langle\langle \mathsf{phd}, \mathsf{college} \rangle\rangle$, illustrated in Figure 4.2(b), contains associations between PhD students in $\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$ and their colleges. Each association in the figure is illustrated as a black double headed arrow in $\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd}, \mathsf{college} \rangle\rangle)$.

◇

Note that the order of the real-world entities in a binary association in the intended domain of an object is insignificant. For example, the intended domain of link schema object $\langle\langle \mathsf{supervisedBy}, \mathsf{phd}, \mathsf{academic} \rangle\rangle$ in S_{Imp} contains the association Nikos Rizopoulos ↔ Peter McBrien. The intended domain of a link schema object $\langle\langle \mathsf{supervises}, \mathsf{academic}, \mathsf{phd} \rangle\rangle$ in S_{Imp} contains the same association in reverse order, *i.e.* Peter McBrien ↔ Nikos Rizopoulos. The order of the real-world entities within these associations is insignificant, therefore the two associations are identical in the real-world and in the intended domains of the two link schema objects.

4.2.2 Semantic Relationships

Now that we have defined the semantics of schema objects based on their intended domains, we can identify semantic relationships between schema objects by perform-

ing a set-based comparison of their intended domains. Our methodology is similar to [59] where *strong* and *weak equivalences* are defined. In [59], the relationships are limited due to the adopted extended ER data model, *e.g.* equivalences between attributes and classes, while we provide a generic framework that can be applied to any data model.

We have defined six types of semantic relationships between schema objects based on whether their intended domains are equivalent, they are subsuming one another, they are intersecting or they are disjoint. For each pair of schema objects $[s_1, s_2]$ only one semantic relationship must be and is applicable. Note that each pair $[s_1, s_2]$ of schema objects is a list of length two and that the order the schema objects appear in this list is significant. We first give the definition of the six types of semantic relationships and then we explain each one.

Definition 4.3. Semantic Relationships The semantic relationship between a pair of schema objects $[s_1, s_2]$ in schemas S_1 and S_2, respectively, must be one of the following:

- **equivalence ($\overset{\mathbf{S}}{=}$)**: We say that s_1 is equivalent to s_2 and we write $\langle s_1, \overset{\mathrm{S}}{=}, s_2 \rangle$, iff

  $\forall \mathrm{Inst}^{int} : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) = \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2)$.

- **subset-subsumption ($\overset{\mathbf{S}}{\subset}$)**: We say that s_1 is subsumed by s_2 and we write $\langle s_1, \overset{\mathrm{S}}{\subset}, s_2 \rangle$, iff

  1. $\forall \mathrm{Inst}^{int} : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) \subseteq \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2)$.
  2. $\neg \langle s_1, \overset{\mathrm{S}}{=}, s_2 \rangle$

- **superset-subsumption ($\overset{\mathbf{S}}{\supset}$)**: We say that s_1 subsumes s_2 and we write $\langle s_1, \overset{\mathrm{S}}{\supset}, s_2 \rangle$, iff

  1. $\forall \mathrm{Inst}^{int} : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) \supseteq \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2)$.

2. $\neg \langle s_1, \overset{\mathrm{S}}{=}, s_2 \rangle$

- **intersection** ($\overset{\mathbf{S}}{\cap}$): We say that s_1 intersects with s_2 and we write $\langle s_1, \overset{\mathrm{S}}{\cap}, s_2 \rangle$, iff

  1. $\forall \mathrm{Inst}^{int} : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) \cap \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2) \neq \emptyset$
  2. $\neg \langle s_1, \overset{\mathrm{S}}{=}, s_2 \rangle$
  3. $\neg \langle s_1, \overset{\mathrm{S}}{\subset}, s_2 \rangle$
  4. $\neg \langle s_1, \overset{\mathrm{S}}{\supset}, s_2 \rangle$
  5. *could exist* $s_3 : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_3) = \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_3) = \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) \cap \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2)$

- **disjointness** ($\overset{\mathbf{S}}{\not\cap}$): We say that s_1 is disjoint with s_2 and we write $\langle s_1, \overset{\mathrm{S}}{\not\cap}, s_2 \rangle$, iff

  1. $\forall \mathrm{Inst}^{int} : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) \cap \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2) = \emptyset$
  2. $\neg\ s_1 \overset{\mathrm{S}}{\subset} s_2$
  3. $\neg\ s_1 \overset{\mathrm{S}}{\supset} s_2$
  4. *could exist* $s_3 : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_3) = \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_3) = \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) \cup \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2)$

- **incompatibility** ($\overset{\mathbf{S}}{\gamma}$): We say that s_1 is incompatible with s_2 and we write $\langle s_1, \overset{\mathrm{S}}{\gamma}, s_2 \rangle$, iff

  1. $\forall \mathrm{Inst}^{int} : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) \cap \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2) = \emptyset$
  2. $\neg\ s_1 \overset{\mathrm{S}}{\subset} s_2$
  3. $\neg\ s_1 \overset{\mathrm{S}}{\supset} s_2$
  4. *could not exist* $s_3 : \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_3) = \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_3) = \mathrm{Dom}^{int}_{S_1,\mathrm{Inst}^{int}}(s_1) \cup \mathrm{Dom}^{int}_{S_2,\mathrm{Inst}^{int}}(s_2)$

□

Intuitively, the semantic relationship between two objects is equivalence ($\overset{S}{=}$) when the intended domains of the objects are equivalent for all possible intended instances, *i.e.* at all times. The semantic relationship between two objects is subset-subsumption ($\overset{S}{\subset}$) when the intended domain of the first schema object is subsumed by the intended domain of the second schema object. However, this condition is true even when the two schema objects are equivalent ($\overset{S}{=}$). Therefore, in the definition of subset-subsumption ($\overset{S}{\subset}$) the second condition that must be satisfied is that the two objects are not equivalent $\neg\langle s_1, \overset{S}{=}, s_2\rangle$. We call tuples such as $\langle s_1, \overset{S}{=}, s_2\rangle$ **semantic mappings**. The symbol of negation, $\neg$, in front of a semantic mapping states that the semantic mapping is not true.

Definition 4.4. Semantic Mapping A **semantic mapping** between two schema objects s_1 and s_2 is the tuple $\langle s_1, rel, s_2\rangle$ where $rel \in \{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}$.

□

The definition of superset-subsumption ($\overset{S}{\supset}$) is is similar to subset-subsumption. Note that if $\langle s_1, \overset{S}{\supset}, s_2\rangle$ then it is also the case that $\langle s_2, \overset{S}{\subset}, s_1\rangle$.

The semantic relationship between two objects is intersection ($\overset{S}{\cap}$) if the intended domains of the two objects are intersecting. The second, third and fourth conditions explicitly prohibit the existence of an equivalence, subset-subsumption and superset-subsumption relationship, respectively, between the objects. The reason behind these restrictions is that the intended domains of equivalent or subsuming schema objects are also intersecting. For example, assume a schema object $\langle\langle \mathsf{imperial\_phd} \rangle\rangle$ that represents all Imperial College PhD students in the AutoMed research group. For all possible $\text{Inst}^{int}$the intended domain $\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{imperial\_phd} \rangle\rangle)$ will be intersecting with $\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$, which represents all PhD students in the AutoMed research group (Example 4.1). In particular, we have that $\forall \text{Inst}^{int}$: $\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{imperial\_phd} \rangle\rangle) \cap \text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle) = \text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{imperial\_phd} \rangle\rangle)$. However, it is also the case that $\forall \text{Inst}^{int} : \text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{imperial\_phd} \rangle\rangle) \subseteq$

$\text{Dom}^{int}_{S_{Imp},\text{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle)$. Thus, even though the intended domains of the two objects are intersecting, we actually have that $\langle \langle\langle \mathsf{imperial\_phd} \rangle\rangle, \overset{\text{S}}{\subset}, \langle\langle \mathsf{phd} \rangle\rangle \rangle$. The fifth condition in intersection checks that the common real-world entities between the two intersecting objects' intended domains constitute a set of elements that a single real-world concept represents. The *could exists* keyword expresses that this concept is relevant to schemas S_1 and S_2 and that s_3 that represents the concept may or may not already appear in S_1 and S_2. Whether the concept is relevant or not is subjective and needs to be confirmed by an expert.

Disjointness, which is the next relationship of our definition, does not have to be explicitly prohibited from the definition of intersection since the first condition of disjointness, which states that the intended domains of the schema object must be disjoint, conflicts with the first condition in the definition of intersection. The same applies for incompatibility. The second and third conditions in disjointness are necessary for the trivial case where either $\text{Dom}^{int}_{S_1,\text{Inst}^{int}}(s_1) = \emptyset$ or $\text{Dom}^{int}_{S_2,\text{Inst}^{int}}(s_2) = \emptyset$. The fourth condition implies that the union of the intended domains of the schema objects is the representation of a real-world concept relevant to S_1 and S_2.

The definition of incompatibility differs with the definition of disjointness only in their fourth condition. The fourth condition of incompatibility states that the union of the intended domains of the schema objects is either not the representation of a real-world concept, or that the real-world concept is not relevant to S_1 and S_2. For example, an expert may decide that $\langle\langle \mathsf{phd} \rangle\rangle$ and $\langle\langle \mathsf{academic} \rangle\rangle$ are disjoint and that the union of the two produces a schema object $\langle\langle \mathsf{phd\_or\_academic} \rangle\rangle$, while another expert may decide that the concept of $\langle\langle \mathsf{phd\_or\_academic} \rangle\rangle$ is not relevant to his/her integration.

In the remaining of this thesis, we are going to use the term **compatible** to describe that two objects are either equivalent, subsuming one another, intersecting or disjoint.

Definition 4.5. Compatibility Two schema objects s_1 and s_2 are said to be **compatible**, $\langle s_1, \overset{S}{\approx}, s_2 \rangle$, iff one of the following conditions is true: (i) $\langle s_1, \overset{S}{=}, s_2 \rangle$, (ii) $\langle s_1, \overset{S}{\subset}, s_2 \rangle$, (iii) $\langle s_1, \overset{S}{\supset}, s_2 \rangle$, (iv) $\langle s_1, \overset{S}{\cap}, s_2 \rangle$, (v) $\langle s_1, \overset{S}{\not\cap}, s_2 \rangle$.

□

Example 4.3. Semantic Mappings Following on from the example in the introduction where schemas S_1^{er} and S_2^{er} about academic papers and academic text books (see Figure 1.3(a)) are matched, a disjointness mapping can be identified for pair $p_1 \equiv [\langle\langle \mathsf{paper} \rangle\rangle, \langle\langle \mathsf{book} \rangle\rangle]$

$$\langle \langle\langle \mathsf{paper} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \mathsf{book} \rangle\rangle \rangle$$

meaning that the two schema objects could be both subsets of a more general object, such as $\langle\langle \mathsf{publication} \rangle\rangle$. Similarly, regarding the two objects' attributes pairs $p_2 \equiv [\langle\langle \mathsf{paper}, \mathsf{title} \rangle\rangle, \langle\langle \mathsf{book}, \mathsf{title} \rangle\rangle]$ and $p_3 \equiv [\langle\langle \mathsf{paper}, \mathsf{year} \rangle\rangle, \langle\langle \mathsf{book}, \mathsf{year} \rangle\rangle]$:

$$\langle \langle\langle \mathsf{paper}, \mathsf{title} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \mathsf{book}, \mathsf{title} \rangle\rangle \rangle$$
$$\langle \langle\langle \mathsf{paper}, \mathsf{year} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \mathsf{book}, \mathsf{year} \rangle\rangle \rangle$$

◇

When matching schemas a list of semantic mappings relating the objects of the schemas is produced. We call this list of semantic mappings, a **schema mapping**.

Definition 4.6. Schema Mapping A **schema mapping** Match is a list of N semantic mappings for the N distinct pairs of objects of a matching task.

□

Automatic schema matching tools that discover schema mappings can never be 100% certain of their correctness. In our schema integration framework we can represent the uncertainty of the schema matching tools by introducing the concepts of **uncertain semantic mapping** and **uncertain schema mapping** in Section 4.3.

4.2.3 Semantic mappings translated to data mappings

Before we examine the representation of uncertainty in schema matching, it is interesting to see what we can say about the extents of a pair of schema objects $[s_1, s_2]$ knowing that its semantic mapping is $\langle s_1, rel_s, s_2 \rangle$. By relating the extents of schema objects we can derive data mappings which are useful during schema merging.

For example, previously we mentioned that the objects $\langle\langle \mathsf{supervisedBy}, \mathsf{phd}, \mathsf{academic} \rangle\rangle$ and $\langle\langle \mathsf{supervises}, \mathsf{academic}, \mathsf{phd} \rangle\rangle$ in S_{Imp} are semantically equivalent even though they specify the same associations of objects in reverse order. By Definition 4.3, we have that $\mathrm{Dom}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{supervisedBy}, \mathsf{phd}, \mathsf{academic} \rangle\rangle) = \mathrm{Dom}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{supervises}, \mathsf{academic}, \mathsf{phd} \rangle\rangle)$. Now since there is a 1:1 mapping between $\mathrm{Ext}^{int}_{S,\mathrm{Inst}^{int}}$ and $\mathrm{Dom}^{int}_{S,\mathrm{Inst}^{int}}$, the above equivalence implies that a 1:1 mapping exists between $\mathrm{Ext}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{supervisedBy}, \mathsf{phd}, \mathsf{academic} \rangle\rangle)$ and $\mathrm{Ext}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{supervises}, \mathsf{academic}, \mathsf{phd} \rangle\rangle)$. Using the function $transpose(\{x, y\}) = \{y, x\}$ which reverses the order of binary tuples we have that

$$\mathrm{Ext}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{supervisedBy}, \mathsf{phd}, \mathsf{academic} \rangle\rangle) = \{transpose(i) \mid i \in \mathrm{Ext}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{supervises}, \mathsf{academic}, \mathsf{phd} \rangle\rangle)\}.$$

Thus, the original equivalence semantic mapping leads to a data mapping between the intended extents of the two objects.

Example 4.4. Translation of intended extents For a more detailed example of how a semantic mapping can be used to determine a data mapping between the intended extents of schema objects consider the following. There is a database at Birkbeck College with schema S_{Bir} that stores information about the members of the AutoMed research group. In particular,

there is an object $\langle\langle \mathsf{member} \rangle\rangle$ with intended extent:

$$\begin{aligned}&\mathrm{Ext}^{int}_{S_{Bir},\mathrm{Inst}^{int}}(\langle\langle \mathsf{member} \rangle\rangle) = \\ &\quad [\{\texttt{Dean Williams}\}, \{\texttt{Hao Fan}\}, \{\texttt{Lucas Zamboulis}\}, \\ &\quad \{\texttt{Nikos Rizopoulos}\}, \{\texttt{Andrew C. Smith}\} \\ &\quad \{\texttt{Alex Poulovassilis}\}, \{\texttt{Peter J. McBrien}\}]\end{aligned}$$

We know that all the PhD students in the AutoMed group, represented by $\langle\langle \mathsf{phd} \rangle\rangle$ in S_{Imp} (Example 4.1), are also members of the group. Thus, we have that $\langle\langle \mathsf{phd} \rangle\rangle, \overset{\mathrm{S}}{\subset}, \langle\langle \mathsf{member} \rangle\rangle\rangle$. This means that $\mathrm{Dom}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle) \subseteq \mathrm{Dom}^{int}_{S_{Bir},\mathrm{Inst}^{int}}(\langle\langle \mathsf{member} \rangle\rangle)$. However, it is not the case that $\mathrm{Ext}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle) \subseteq \mathrm{Ext}^{int}_{S_{Bir},\mathrm{Inst}^{int}}(\langle\langle \mathsf{member} \rangle\rangle)$, due to the fact that in Birkbeck the encoding of the AutoMed members includes their middle name initials.

The function *removeInitials* can translate between the intentional extents of $\langle\langle \mathsf{member} \rangle\rangle$ and $\langle\langle \mathsf{phd} \rangle\rangle$ by removing the middle name initials from each instance of $\langle\langle \mathsf{member} \rangle\rangle$, if it has any, *i.e.*

$$\begin{aligned}\{removeInitials(i) \mid \; & i \in \mathrm{Ext}^{int}_{S_{Bir},\mathrm{Inst}^{int}}(\langle\langle \mathsf{member} \rangle\rangle)\} = \\ & \{\{\texttt{Dean Williams}\}, \{\texttt{Hao Fan}\}, \{\texttt{Lucas Zamboulis}\}, \\ & \{\texttt{Nikos Rizopoulos}\}, \{\texttt{Andrew Smith}\}, \\ & \{\texttt{Alex Poulovassilis}\}, \{\texttt{Peter McBrien}\}\}\end{aligned}$$

Based on the *removeInitials* function and the fact that $\langle\langle \mathsf{phd} \rangle\rangle$ is subsumed by $\langle\langle \mathsf{member} \rangle\rangle$, we have that the intended extent of $\langle\langle \mathsf{phd} \rangle\rangle$ is subsumed by the translated intended extent of $\langle\langle \mathsf{member} \rangle\rangle$: $\mathrm{Ext}^{int}_{S_{Imp},\mathrm{Inst}^{int}}(\langle\langle \mathsf{phd} \rangle\rangle) \subseteq \{removeInitials(i) \mid i \in \mathrm{Ext}^{int}_{S_{Bir},\mathrm{Inst}^{int}}(\langle\langle \mathsf{member} \rangle\rangle)\}$.

◇

In general, we see that the semantic mapping between two schema objects implies a data mapping between the intended extents of the objects. The following proposition

specifies the data mapping derived from each distinct compatibility mapping.

Proposition 4.1. Based on Definition 4.3, a function f that can translate between the intended extents of equivalent, subsuming, intersecting and disjoint schema objects must have the following properties:

- $\langle s_1, \overset{S}{=}, s_2 \rangle$,
  $\exists f : \forall \text{Inst}^{int} (\text{Ext}^{int}_{S_1,\text{Inst}^{int}}(s_1) = \{f(i) \mid i \in \text{Ext}^{int}_{S_2,\text{Inst}^{int}}(s_2)\})$.

- $(\langle s_1, \overset{S}{\subset}, s_2 \rangle$,
  $\exists f : \forall \text{Inst}^{int} (\text{Ext}^{int}_{S_1,\text{Inst}^{int}}(s_1) \subseteq \{f(i) \mid i \in \text{Ext}^{int}_{S_2,\text{Inst}^{int}}(s_2)\})$.

- $(\langle s_1, \overset{S}{\supset}, s_2 \rangle$,
  $\exists f : \forall \text{Inst}^{int} (\text{Ext}^{int}_{S_1,\text{Inst}^{int}}(s_1) \supseteq \{f(i) \mid i \in \text{Ext}^{int}_{S_2,\text{Inst}^{int}}(s_2)\})$.

- $(\langle s_1, \overset{S}{\cap}, s_2 \rangle$,
  $\exists f : \forall \text{Inst}^{int} (\text{Ext}^{int}_{S_1,\text{Inst}^{int}}(s_1) \cap \{f(i) \mid i \in \text{Ext}^{int}_{S_2,\text{Inst}^{int}}(s_2)\} \neq \emptyset)$.

- $(\langle s_1, \overset{S}{\not\cap}, s_2 \rangle$,
  $\exists f : \forall \text{Inst}^{int}, \exists s_3 (\text{Ext}^{int}_{S_{12},\text{Inst}^{int}}(s_3) = \text{Ext}^{int}_{S_1,\text{Inst}^{int}}(s_1) \cup \{f(i) \mid i \in \text{Ext}^{int}_{S_2,\text{Inst}^{int}}(s_2)\})$
  and $\text{Ext}^{int}_{S_1,\text{Inst}^{int}}(s_1) \cap \{f(i) \mid i \in \text{Ext}^{int}_{S_2,\text{Inst}^{int}}(s_2)\} = \emptyset$.

In general, identifying such a function f is a very hard problem. The number of such functions is infinite and thus the problem is intractable. It is essentially the same problem as the identification of data mappings during schema matching (see Section 2.2). As far as we know only two existing schema matching approach, [31, 112] (reviewed in Section 2.2.2), could possibly be used to identify such a function f. In these approaches, unit conversion data mappings are derived, *e.g.* "weight-kg $= 2.2 *$ weight-pounds", which are appropriate candidates for function f. In this particular example, the intended extents of the semantically equivalent schema objects weight-kg and weight-pounds can be translated using the function $f(x) = 2.2 * x$.

4.3 Uncertain Semantic Mappings

Based on the definitions in the previous section, we can identify a semantic mapping between two schema objects if we know the intended domains of the objects and we are certain about them. However, intended domains are not available to an automatic matching tool. Therefore, the matching tool will produce semantic mappings with a high degree of uncertainty. In this section, we are going to explain our methodology for the representation of uncertainty in the schema matching process.

To deal with schema matching uncertainty, we have adopted **Shafer's theory of belief functions** [97], also known as **evidence theory**. The theory deals with a so-called **frame of discernment**, which is the set of all possible elementary events, and is represented by the letter Θ. In our framework where the uncertainty lies in the exact semantic relationship in a semantic mapping, the elementary events are all semantic relationships defined in the previous section, *i.e.* $\Theta = \Theta_{rel} = \{\overset{s}{=}, \overset{s}{\cap}, \overset{s}{\subset}, \overset{s}{\supset}, \overset{s}{\not\cap}, \overset{s}{\not\sim}\}$. Based on Shafer's theory, if we are uncertain about which single semantic relationship is the correct one for a given pair, we can assign probability masses to several distinct sets $A_i \subseteq \Theta_{rel}$, of semantic relationships.

The basis of the measure of uncertainty in Shafer's theory is a function called **basic probability assignment** (BPA) that assigns some probability mass to each element of 2^{Θ}:

Definition 4.7. Basic Probability Assignment (BPA) A function $m : 2^{\Theta} \rightarrow [0, 1]$ is called *basic probability assignment* whenever:

- $m(\emptyset) = 0$
- $\sum_{A \subseteq \Theta} m(A) = 1$

□

The above definition states that no probability mass is assigned to the impossible event $\emptyset$, *i.e.* the empty set, and that the total probability mass that can be distributed has measure one.

From a BPA function, we can compute the belief and plausibility of any subset A of Θ. The belief $Bel(A)$ assigned to a set A is the sum of all probability masses assigned to any evidence set B that states that the set A is true, *i.e.* $B \subseteq A$. The plausibility $Pl(A)$ assigned to a set A is the sum of all probability masses assigned to any evidence set B that is consistent with set A, *i.e.* $B \cap A \neq \emptyset$.

Definition 4.8. Belief and Plausibility

$$\begin{aligned} \mathrm{Bel}(A) &= \sum_{B \subseteq A} m(B) \\ \mathrm{Pl}(A) &= \sum_{B \subseteq \Theta, B \cap A \neq \emptyset} m(B) \end{aligned}$$

□

Since for each set B, with $B \subseteq A$, the statement $B \cap A \neq \emptyset$ is true, we have that $Bel(A) \leq Pl(A)$. Thus, a BPA defines a certainty interval $[Bel(A), Pl(A)]$ on each set $A \subseteq \Theta$.

Example 4.5. Belief and Plausibility In our schema integration framework, the set $\{\overset{\mathrm{S}}{=}, \overset{\mathrm{S}}{\not=}\}$ represents the event "The correct semantic relationship is either equivalence or incompatibility", and $m(\{\overset{\mathrm{S}}{=}, \overset{\mathrm{S}}{\not=}\})$ is the probability mass supporting exactly this event.

If some probability mass is assigned to the set $\{\overset{\mathrm{S}}{=}\}$, this increases our belief in all the events containing it. In fact, if we have some evidence supporting the event "The true semantic relationship is equivalence", the same evidence

increases also our belief in the event "The true semantic relationship is either equivalence or incompatibility".

Plausibility on $\{\overset{S}{=}, \overset{S}{\not\sim}\}$ is the sum of all probability masses that are consistent with $\{\overset{S}{=}, \overset{S}{\not\sim}\}$. For example, some probability mass assigned to $\{\overset{S}{=}, \overset{S}{\not\cap}\}$ tells us that $\{\overset{S}{=}, \overset{S}{\not\sim}\}$ is plausible, without increasing our belief in it, because the correct relationship could be disjointness.

◇

Based on Shafer's theory, we can formally define uncertain semantic mappings:

Definition 4.9. Uncertain Semantic Mapping (USM) An *uncertain semantic mapping* between two schema objects s_1 and s_2 is the tuple $\langle s_1, m, s_2 \rangle$ where m is a BPA on $\Theta_{rel} = \{\overset{S}{=}, \overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}$.

□

Automatic matching tools can explicitly represent their uncertainty on a matching task by producing a list of USMs. We call this list of USMs an **uncertain schema mapping**.

Definition 4.10. Uncertain Schema Mapping An **uncertain schema mapping** u-Match is a list of N USMs for the N distinct pairs of objects of a matching task.

□

4.4 Aggregation of Uncertain Semantic Mappings

USMs can be aggregated using Dempster's combination rule [97]. The rule takes two BPAs m_1 and m_2 over the same frame of discernment Θ as input and produces the probability mass for a subset A of Θ.

Intuitively, a given subset A of Θ may be the intersection of several different pairs A_i, A_j. In order to obtain the total mass assigned exactly to A, the sum of all products $m_1(A_i)m_2(A_j)$, $A_i \cap A_j = A$, needs to be calculated. However, if there are assignments $m_1(A_i)$, $m_2(A_j)$ such that $A_i \cap A_j = \emptyset$ and $m_1(A_i)m_2(A_j) > 0$, *i.e.* some probability mass greater than zero is assigned to the empty set, then the combination $m_1 \oplus m_2$ of m_1 and m_2 is not a BPA according to Definition 4.7. Therefore, in order to make $m_1 \oplus m_2$ a BPA, we need to exclude cases such as $A_i \cap A_j = \emptyset$. This can be achieved by normalizing all masses $m_1 \oplus m_2(A)$ and dividing them with the sum of all products $m_1(A_i)m_2(A_j)$ for which $A_i \cap A_j \neq \emptyset$,

$$\sum_{A_i \subseteq \Theta, A_j \subseteq \Theta, A_i \cap A_j \neq \emptyset} m_1(A_i)m_2(A_j)$$

Thus, the rule that combines two BPAs m_1 and m_2 is the following:

$$m_1 \oplus m_2(A) = \begin{cases} 0 & \text{if } A = \emptyset \\ \frac{\sum_{A_i \subseteq \Theta, A_j \subseteq \Theta, A_i \cap A_j = A} m_1(A_i)m_2(A_j)}{\sum_{A_i \subseteq \Theta, A_j \subseteq \Theta, A_i \cap A_j \neq \emptyset} m_1(A_i)m_2(A_j)} & \text{if } A \neq \emptyset \end{cases} \tag{4.1}$$

Based on Dempster's combination rule, the aggregation of a pair of uncertain semantic mappings $\langle s_1, m_1, s_2 \rangle$ and $\langle s_1, m_2, s_2 \rangle$ between schema objects s_1 and s_2 is a new uncertain semantic mapping $\langle s_1, m_{1\oplus 2}, s_2 \rangle$ where $m_{1\oplus 2}$ is a BPA produced using Dempster's combination rule on m_1 and m_2 for every subset A of Θ_{rel}, *i.e.* $\forall A \subseteq \Theta_{rel} : m_{1\oplus 2}(A) = m_1 \oplus m_2(A)$.

Dempster's combination rule is commutative and associative, thus the aggregation of N uncertain semantic mappings is obtained by iteratively aggregating pairs of uncertain semantic mappings $N - 1$ times. For example, if there was an additional uncertain semantic mapping $\langle s_1, m_3, s_2 \rangle$, then the aggregation of the three mappings would be a new mapping $\langle s_1, m_{1\oplus 2\oplus 3}, s_2 \rangle$ where $m_{1\oplus 2\oplus 3}$ is a BPA produced using Dempster's combination rule on $m_{1\oplus 2}$ and m_3 for every subset A of Θ_{rel}, *i.e.* $\forall A \subseteq \Theta_{rel} : m_{1\oplus 2\oplus 3}(A) = m_{1\oplus 2} \oplus m_3(A)$.

Example 4.6. Aggregation of USMs Assume that we have three different experts, E_1, E_2 and E_3, matching the entities paper in S_1^{er} and book S_2^{er} and their attributes. Remember that paper represents academic paper publications and book represents academic text book publications.

Expert E_1 compares the cardinalities of the schema objects, *i.e.* the number of their instances. The expert takes the view that if the cardinalities are equal, subsumption is not possible. If the cardinality of the first object is greater than the other, then they cannot be equivalent and the second object cannot subsume the first one. Notice that this expert assumes that all instances belonging to those objects in the real world are stored in the database. It would be possible to improve the expert so that some instances can be missing, using fuzzy comparisons. The cardinality of $\langle\langle \mathsf{paper} \rangle\rangle$ is much greater than that of $\langle\langle \mathsf{book} \rangle\rangle$, because there are many more research papers than text books. Therefore, the first expert can exclude equivalence and subset-subsumption. The USM produced by this expert is:

$$m_{E_1}(\{\overset{\mathrm{S}}{\supset}, \overset{\mathrm{S}}{\cap}, \overset{\mathrm{S}}{\not\cap}, \overset{\mathrm{S}}{\not\sim}\}) = 1$$
$$\langle \langle\langle \mathsf{paper} \rangle\rangle, m_{E_1}, \langle\langle \mathsf{book} \rangle\rangle \rangle$$

Expert E_2 uses schema structure and object names to perform matching. The expert identifies that both $\langle\langle \mathsf{paper} \rangle\rangle$ and $\langle\langle \mathsf{book} \rangle\rangle$ have two attributes title and year. However, they have different names and their identifiers, bibtex and id, do not seem compatible. Thus, the expert is uncertain about the compatibility of $\langle\langle \mathsf{paper} \rangle\rangle$ and $\langle\langle \mathsf{book} \rangle\rangle$ and produces the following mapping:

$$m_{E_2}(\{\overset{\mathrm{S}}{=}, \overset{\mathrm{S}}{\subset}, \overset{\mathrm{S}}{\supset}, \overset{\mathrm{S}}{\cap}, \overset{\mathrm{S}}{\not\cap}\}) = .4, m_{E_2}(\{\overset{\mathrm{S}}{\not\sim}\}) = .6$$
$$\langle \langle\langle \mathsf{paper} \rangle\rangle, m_{E_2}, \langle\langle \mathsf{book} \rangle\rangle \rangle$$

Expert E_3 compares the instances of two schema objects. For efficiency reasons, it only compares a sample of the instances of $\langle\langle \mathsf{paper} \rangle\rangle$ with all

the instances of the $\langle\langle \mathsf{book} \rangle\rangle$, and *vice versa*. This induces uncertainty on the result. In our example, the expert E_3 cannot find matches between the instances of the two objects, because a publication cannot be both a paper and a book. Therefore, it will support the set of relationships $\{\overset{S}{\not\equiv}, \overset{S}{\not\sim}\}$. However, as already said, the expert cannot be certain of this information. Its USM is :

$$m_{E_3}(\{\overset{S}{\not\equiv}, \overset{S}{\not\sim}\}) = .8, m_{E_3}(\Theta_{rel}) = .2$$
$$\langle \langle\langle \mathsf{paper} \rangle\rangle, m_{E_3}, \langle\langle \mathsf{book} \rangle\rangle \rangle$$

Experts E_1 and E_3 think that disjointness and incompatibility are plausible and expert E_2 believes in incompatibility, ranking it higher than any other relationship. Combining the three BPAs m_{E_1}, m_{E_2}, and m_{E_3} that the three experts produce, gives the BPA m_{p_1} as follows.

First m_{E_1} and m_{E_2} must be combined. The two BPAs intersect in these two sets only:

- $\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}, \overset{S}{\not\sim}\} \cap \{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}\} = \{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}\}$:
  $m_{E_1}(\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}, \overset{S}{\not\sim}\}) \times m_{E_2}(\{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}\}) = 1 \times 0.4 = 0.4$
- $\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}, \overset{S}{\not\sim}\} \cap \{\overset{S}{\not\sim}\} = \{\overset{S}{\not\sim}\}$:
  $m_{E_1}(\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}, \overset{S}{\not\sim}\}) \times m_{E_2}(\{\overset{S}{\not\sim}\}) = 1 \times 0.6 = 0.6$

Based on Equation 4.1, we have

$m_{E_1 \oplus E_2}(\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}\}) = 0.4/(0.4 + 0.6) = 0.4$

$m_{E_1 \oplus E_2}(\{\overset{S}{\not\sim}\}) = 0.6/(0.4 + 0.6) = 0.6$

Now, m_{p_1} can be computed by combining $m_{E_1 \oplus E_2}$ and m_{E_3}. These two BPAs intersect in the following sets only:

- $\{\overset{S}{\not\equiv}\}$: is the intersection of $\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}\}$ of $m_{E_1 \oplus E_2}$ with $\{\overset{S}{\not\equiv}, \overset{S}{\not\sim}\}$ of m_{E_3}, thus
  $m_{E_1 \oplus E_2}(\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\equiv}\}) \times m_{E_3}(\{\overset{S}{\not\equiv}, \overset{S}{\not\sim}\}) = 0.4 \times 0.8 = 0.32$

- $\{\overset{S}{\not\sim}\}$: is the intersection of $\{\overset{S}{\not\sim}\}$ of $m_{E_1 \oplus E_2}$ with $\{\overset{S}{\not\cap}, \overset{S}{\not\sim}\}$ of m_{E_3} and the intersection of $\{\overset{S}{\not\sim}\}$ of $m_{E_1 \oplus E_2}$ with Θ_{rel} of m_{E_3}, thus
  $m_{E_1 \oplus E_2}(\{\overset{S}{\not\sim}\} \times m_{E_3}(\{\overset{S}{\not\cap}, \overset{S}{\not\sim}\}) + m_{E_1 \oplus E_2}(\{\overset{S}{\not\sim}\}) \times m_{E_3}(\Theta_{rel}) = 0.6 \times 0.8 + 0.6 \times 0.2 = 0.6$
- $\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}$: is the intersection of $\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}$ of $m_{E_1 \oplus E_2}$ and Θ_{rel} of m_{E_3}, thus
  $m_{E_1 \oplus E_2}(\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}) \times m_{E_3}(\Theta_{rel}) = 0.4 \times 0.2 = 0.08$

Based on Equation 4.1, we have

$m_{p_1}(\{\overset{S}{\not\cap}\}) = m_{E_1 \oplus E_2 \oplus E_3}(\{\overset{S}{\not\cap}\}) = 0.32/(0.32 + 0.6 + 0.08) = 0.32$

$m_{p_1}(\{\overset{S}{\not\sim}\}) = m_{E_1 \oplus E_2 \oplus E_3}(\{\overset{S}{\not\sim}\}) = 0.6/(0.32 + 0.6 + 0.08) = 0.6$

$m_{p_1}(\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}) = m_{E_1 \oplus E_2 \oplus E_3}(\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}) = 0.08/(0.32 + 0.6 + 0.08) = 0.08$

Thus, the USM for p_1 is the following:

$$m_{p_1}(\{\overset{S}{\not\sim}\}) = .60, m_{p_1}(\{\overset{S}{\not\cap}\}) = .32, m_{p_1}(\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}) = .08$$
$$\langle \langle\!\langle \mathsf{paper} \rangle\!\rangle, m_{p_1}, \langle\!\langle \mathsf{book} \rangle\!\rangle \rangle \tag{4.2}$$

The same three experts matching the attributes of $\langle\!\langle \mathsf{paper} \rangle\!\rangle$ and $\langle\!\langle \mathsf{book} \rangle\!\rangle$ would produce:

$$m'_{E_1}(\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}) = 1$$
$$\langle \langle\!\langle \mathsf{paper}, \mathsf{title} \rangle\!\rangle, m'_{E_1}, \langle\!\langle \mathsf{book}, \mathsf{title} \rangle\!\rangle \rangle$$
$$m'_{E_2}(\{\overset{S}{\not=}, \overset{S}{\not\subset}, \overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}) = .8, m'_{E_2}(\{\overset{S}{\not\sim}\}) = .2$$
$$\langle \langle\!\langle \mathsf{paper}, \mathsf{title} \rangle\!\rangle, m'_{E_2}, \langle\!\langle \mathsf{book}, \mathsf{title} \rangle\!\rangle \rangle$$
$$m'_{E_3}(\{\overset{S}{\not\cap}, \overset{S}{\not\sim}\}) = .8, m'_{E_3}(\Theta_{rel}) = .2$$
$$\langle \langle\!\langle \mathsf{paper}, \mathsf{title} \rangle\!\rangle, m'_{E_3}, \langle\!\langle \mathsf{book}, \mathsf{title} \rangle\!\rangle \rangle$$

which aggregated give the following USM:

$$m_{p_2}(\{\overset{S}{\not\sim}\}) = .20, m_{p_2}(\{\overset{S}{\not\cap}\}) = .64, m_{p_2}(\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}) = .16$$
$$\langle \langle\!\langle \mathsf{paper}, \mathsf{title} \rangle\!\rangle, m_{p_2}, \langle\!\langle \mathsf{book}, \mathsf{title} \rangle\!\rangle \rangle$$

	p_1		p_2		p_3	
	[⟪paper⟫,⟪book⟫]		[⟪paper, title⟫,⟪book, title⟫]		[⟪paper, year⟫,⟪book, year⟫]	
	Bl	Pl	Bl	Pl	Bl	Pl
$\{\overset{S}{=}\}$	0	0	0	0	0	0
$\{\overset{S}{\subset}\}$	0	0	0	0	0	0
$\{\overset{S}{\supset}\}$	0	0.08	0	0.16	0	0.16
$\{\overset{S}{\cap}\}$	0	0.08	0	0.16	0	0.16
$\{\overset{S}{\not\cap}\}$	0.32	0.4	0.64	0.8	0.64	0.8
$\{\overset{S}{\not\sim}\}$	0.6	0.6	0.2	0.2	0.2	0.2

Table 4.4: Belief, plausibility of alternative semantic relationships between pairs p_1, p_2 and p_3

The BPA m_{p_2} is the BPA $m'_{E_1 \oplus E_2 \oplus E_3}$. To compute it, first we see that m'_{E_1} and m'_{E_2} are intersecting only in: $\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}$ and $\{\overset{S}{\not\sim}\}$. Thus,

$m'_{E_1 \oplus E_2}(\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}) = (1 \times 0.8)/(1 \times 0.8 + 1 \times 0.2) = 0.8$

$m'_{E_1 \oplus E_2}(\{\overset{S}{\not\sim}\}) = (1 \times 0.2)/(1 \times 0.8 + 1 \times 0.2) = 0.2$

Now, $m'_{E_1 \oplus E_2}$ and m'_{E_3} are intersecting in sets: $\{\overset{S}{\not\cap}\}$, $\{\overset{S}{\not\sim}\}$ and $\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}$. As previously, we have

$m_{p_2}(\{\overset{S}{\not\cap}\}) = m'_{E_1 \oplus E_2 \oplus E_3}(\{\overset{S}{\not\cap}\}) = \frac{0.8 \times 0.8}{0.8 \times 0.8 + (0.2 \times 0.8 + 0.2 \times 0.2) + 0.8 \times 0.2} = 0.64$

$m_{p_2}(\{\overset{S}{\not\sim}\}) = m'_{E_1 \oplus E_2 \oplus E_3}(\{\overset{S}{\not\sim}\}) = \frac{0.2 \times 0.8 + 0.2 \times 0.2}{0.8 \times 0.8 + (0.2 \times 0.8 + 0.2 \times 0.2) + 0.8 \times 0.2} = 0.20$

$m_{p_2}(\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}) = m'_{E_1 \oplus E_2 \oplus E_3}(\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}) = \frac{0.8 \times 0.2}{0.8 \times 0.8 + (0.2 \times 0.8 + 0.2 \times 0.2) + 0.8 \times 0.2} = 0.16$

For the pair ⟪paper, year⟫ and ⟪book, year⟫, the experts produce USMs which have the same BPAs m'_{E_1}, m'_{E_2} and m'_{E_3} as above. Thus, a similar mapping would be produced for this pair :

$$m_{p_3}(\{\overset{S}{\not\sim}\}) = .20, m_{p_3}(\{\overset{S}{\not\cap}\}) = .64, m_{p_3}(\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}) = .16$$
$$\langle \text{⟪paper, year⟫}, m_{p_3}, \text{⟪book, year⟫} \rangle$$

while we can assume in this example that the remaining attribute pairs of paper and book give certain incompatibility mappings.

In Table 4.4 we have computed the belief and plausibility of every alternative semantic relationship for the three pairs.

◇

Dempster's combination rule can be applied to combine two *independent* BPAs. This means that the information used to derive each BPA must be independent. In our previous example (Example 4.6) for instance, the three experts produce USMs (each USM is associated with a BPA) based on completely different information to compare schema objects (cardinalities, structure and instances). Thus, in Example 4.6, we *can* apply Dempter's rule to combine the experts' BPAs. If, instead, two experts both produced BPAs by comparing the instances of the schema objects and the third expert produced BPAs by comparing the cardinalities of the schema objects, then the final combined BPA would be biased towards the results of the first two experts, *i.e.* biased towards the cardinality information.

Other existing schema matching approaches, *e.g.* [81], also have to deal with this issue, which is usually resolved by assigning weights to the experts. The way these weights are derived though is not trivial. In some approaches, *e.g.* [33], these weights are assigned arbitrarily, while in [35] there is a training phase where the weights are learned based on data from previous manual matching tasks.

We currently require our experts to be independent. This implies that we cannot use experts as black boxes, but we need to know the information they use to perform matching. If there are experts that use dependent information to produce BPAs then these experts have to be merged into a single expert first, in order to be incorporated into our framework. As future work, we could investigate extensions of Dempster's combination rule, *e.g.* the *tradeoff* method [97], that essentially assign weights to the experts.

4.5 Supported Features

We mentioned in the introduction of this chapter a list of features that we require our approach to the representation of schema matching uncertainty to support. In this section, we illustrate how we support these features.

- ranking: USMs can be used to rank the semantic relationships for each pair of schema objects based on the belief or the plausibility assigned to each semantic relationship. For example, based on the plausibility values in Table 4.4, the semantic relationships for pair p_1 have the following ranks: (1) $\overset{S}{\not\supset}$; (2) $\overset{S}{\not\cap}$; (3) $\overset{S}{\supset}$, $\overset{S}{\cap}$; (4) $\overset{S}{=}$, $\overset{S}{\subset}$.

- degree of uncertainty: each semantic relationship for each pair of schema objects is assigned a belief and a plausibility value based on USMs. Belief and plausibility can be used to show the degree of uncertainty of each semantic relationship. For example, for pair p_1 in Table 4.4, we have that semantic relationship $\overset{S}{\not\supset}$ is assigned a plausibility value of 0.6.

- ignorance: USMs can be used to show total and partial ignorance about the possible semantic relationships for each pair of objects. USMs can express total ignorance by assigning all their probability mass to the set of all semantic relationships, $m(\Theta_{rel}) = 1$. USMs can also express partial ignorance by assigning a part of their probability mass to Θ_{rel}. For example, a USM with $m(\{\overset{S}{=}\}) = .2, m(\Theta_{rel}) = .8$ means that we have some evidence that the two objects are equivalent, but we are not sure.

- certainty: A USM that assigns all its probability mass to a single semantic relationship for a pair of objects, *e.g.* $m(\{\overset{S}{=}\}) = 1$, expresses its certainty about the particular semantic relationship for that pair.

4.6 Comparison with other approaches

There are several approaches to modelling uncertainty [84] that could also be used potentially to represent schema matching uncertainty. In this section, we compare Dempster-Shafer's theory (adopted in our approach) against the numerical formalisms (probabilities, fuzzy sets) reviewed in [84], and against rough sets.

Dempster-Shafer's theory is known to be a generalization of Bayesian probability theory [84]. Thus, using probabilities to represent schema matching uncertainty is subsumed by our approach. Specifically, USMs could be used to assign probabilities, if probability mass is assigned to singleton sets of semantic relationships. For example, a USM with $m(\{\overset{S}{\not\sqcap}\}) = .4, m(\{\overset{S}{\not\sim}\}) = .6$ means that for the particular pair the probability of disjointness is .4, while the probability of incompatibility is .6.

Fuzzy set theory [113] is another approach that could be used to represent schema matching uncertainty. A fuzzy set is specified by a tuple $\langle S, g_S \rangle$ where S is a set and g_S is a function with range $[0, 1]$. For each x, $g_S(x)$ is the *grade* of membership of x in S, *i.e.* $g_S(x)$ shows the degree of uncertainty whether x is a member of S or not. Fuzzy set theory provides support for two of the features we require in the representation of schema matching uncertainty: certainty and ignorance. For example, if $S_{\{\overset{S}{=},\overset{S}{\subset},\overset{S}{\supset},\overset{S}{\sqcap},\overset{S}{\not\sqcap}\}}$ is the set of all compatible pairs of objects, then the fact that p_1 is definitely a compatible pair is expressed with $g_{\{\overset{S}{=},\overset{S}{\subset},\overset{S}{\supset},\overset{S}{\sqcap},\overset{S}{\not\sqcap}\}}(p_1) = 1$. If we are uncertain whether p_2 is a compatible or an incompatible pair, we can assign $g_{\{\overset{S}{=},\overset{S}{\subset},\overset{S}{\supset},\overset{S}{\sqcap},\overset{S}{\not\sqcap},\overset{S}{\not\sim}\}} = 1$ showing total ignorance. However, fuzzy set theory does not fully support our other two feature requirements: ranking and assignment for a degree of uncertainty to each semantic relationship. We illustrate this with the following example.

Assume the following grades of membership: $g_{\{\overset{S}{\supset},\overset{S}{\subset},\overset{S}{\sqcap}\}} = 0.2$, $g_{\{\overset{S}{\supset},\overset{S}{\sqcap},\overset{S}{\not\sqcap}\}} = 0.2$, $g_{\{\overset{S}{=},\overset{S}{\sqcap}\}} = 0.5$ and $g_{\{\overset{S}{\not\sim}\}} = 0.1$. Based on these grades, fuzzy set theory does not provide a way to derive the rank of each semantic relationship nor its degree of uncertainty.

In the example, it is more likely that $\overset{S}{\cap}$ should be ranked first because it is supported by three grades, but this cannot be derived formally. Additionally, it is not obvious at which position $\overset{S}{\not\sim}$ should be ranked. Using Dempster-Shafer's theory, we can use belief and plausibility to rank each semantic relationship. In this particular example, if we use plausibility to rank the relationships we would have: $Pl(\overset{S}{\cap}) = 1, Pl(\overset{S}{=}) = 0.5, Pl(\overset{S}{\supset}) = 0.4, Pl(\overset{S}{\subset}) = Pl(\overset{S}{\not\cap}) = 0.2, Pl(\overset{S}{\not\sim}) = 0.1$.

Finally, rough set theory [85] is another approach that could also be used to represent schema matching uncertainty. A rough set is an approximation of a set specified by lower and upper bounds. For example, in our setting, a rough set of all compatible pairs of schema objects in a matching task could be specified by the tuple $\langle\{p_1\}, \{p_1, p_2, p_3\}\rangle$, which states that the set of compatible pairs of objects is a superset of $\{p_1\}$ and a subset of set $\{p_1, p_2, p_3\}$, *i.e.* we are certain that p_1 is a compatible pair but we are uncertain about p_2 and p_3. Thus, rough sets can either be used to show certainty about the semantic relationship of a pair, *e.g.* p_1, or total ignorance, *e.g.* p_2 and p_3. Certainty and total ignorance are special cases of the schema matching uncertainty that can be represented by Dempster-Shafer's theory. Regarding the previous example of pairs p_1, p_2, p_3, the certainty that p_1 is a compatible pair can be represented in Dempster-Shafer's theory with $m(\{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}) = 1$, while for pairs p_2, p_3, which we are uncertain whether they compatible or not, we can state that $m(\{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}) = 1$. Thus, regarding schema matching uncertainty, Dempster-Shafer's theory is more expressive than rough set theory.

4.7 Top-K Schema Matching

Existing schema matching approaches produce a single schema mapping (Definition 4.6) for each matching task. For example, in COMA++ [5] the multiple correspondences produced by matching constitute a single schema mapping. If uncertainty is exposed, then an uncertain schema mapping (Definition 4.10) is produced.

The uncertain schema mapping assigns a probability mass to several distinct sets of schema mappings.

Assuming that a task is composed of N pairs of objects and that matching seeks for the six semantic relationships of Definition 4.3 then there are 6^N possible schema mappings. Most existing schema matching approaches only consider two semantic relationships, compatibility and incompatibility, therefore 2^N schema mappings are possible, and they return one of these possible schema mappings as the final result.

If we are uncertain about the correct schema mapping, we can assign probability masses to several distinct sets of schema mappings. Using Shafer's theory in this setting, the frame of discernment is the set of all 6^N possible schema mappings, *i.e.* $\Theta_{Match} = \{\text{Match}_1, \ldots, \text{Match}_{6^N}\}$, where Match_i is a schema mapping. From an uncertain schema mapping a BPA M_{Match} on Θ_{Match} can be derived. The derivation of M_{Match} is described next.

An uncertain schema mapping defines N USMs for N pairs of objects. Each USM specifies a BPA $m_{p_i}, 1 \leq i \leq N$, on Θ_{rel} for each pair p_i. Assume that m_{p_i} assigns probability masses to sets $rels_{ij} \subseteq \Theta_{rel}$. For each set $rels_{ij}$, there is a set of schema mappings $S_{ij} \subseteq \Theta_{Match}$, with each schema mapping in S_{ij} specifying relationships for pair p_i only from $rels_{ij}$, *i.e.*

$$\forall m_i, \forall \text{Match}_k \in S_{ij}, \text{Match}_k[i] \in rels_{ij} \tag{4.3}$$

where $\text{Match}_k[i]$ is the relationship for pair p_i in schema mapping Match_k. Each m_i on Θ_{rel} can now be translated into a BPA M_i on Θ_{Match}, with $m_i(rels_{ij}) = M_i(S_{ij})$. The final BPA M_{Match} is produced by combining the M_i's on Θ_{Match} using Dempster's combination rule, *i.e.* $M_{Match} = M_1 \oplus \ldots \oplus M_N$.

Example 4.7. Assigning probability mass to schema mappings Following on from Example 4.6, the matching task of comparing the entity paper and its attributes (6 objects) against the entity book and its attributes (5 objects) contains 30 pairs of objects, $N = 30$. Assuming that we are interested in 6 relationships, the frame of discernment Θ_{Match} for the uncertain schema mapping has $30^6 = 729,000,000$ possible schema mappings.

Following on from Example 4.6, the USMs for the three pairs p_1, p_2 and p_3 allow four semantic relationships for each pair. For the remaining 27 pairs, $p_4, \ldots, p_{30}$, their USMs have BPA $m_{p_j}(\{\overset{S}{\not\sim}\}) = 1$, $4 \leq j \leq 30$, *i.e.* the tool is certain that the remaining 27 pairs are incompatible. The three pairs, each one with four possible semantic relationships, specify $4^3 = 64$ possible schema mappings. In these 64 schema mappings, pairs $p_4, \ldots, p_{30}$ are all incompatible. The remaining $30^6 - 4^3$ schema mappings are not possible since they assign to pairs $p_4, \ldots, p_{30}$ a compatibility relationship. Table 4.5 shows the first 16 possible schema mappings focusing on pairs p_1, p_2 and p_3, and abbreviating the remaining 27 pairs whose only configuration is the incompatibility relationship.

To derive M_{Match}, we examine BPAs m_{p_1}, m_{p_2} and m_{p_3}. The BPA m_{p_1} assigns probability mass to sets $\{\overset{S}{\not\sim}\}$, $\{\overset{S}{\not\cap}\}$ and $\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}$ for pair p_1. In the 64 possible schema mappings, p_1 has relationship $\overset{S}{\not\sim}$ in mappings $\text{Match}_1, \ldots,$ Match_{16}, *i.e.* the set $\{\overset{S}{\not\sim}\}$ for p_1 corresponds to set $S_{11} = \{\text{Match}_1, \ldots, \text{Match}_{16}\}$. Similarly, $\{\overset{S}{\not\cap}\}$ for p_1 appears in schema mappings $S_{12} = \{\text{Match}_{17}, \ldots, \text{Match}_{32}\}$ and $\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}$ appear in $S_{13} = \{\text{Match}_{17}, \ldots, \text{Match}_{64}\}$. Thus, m_{p_1} is translated into M_1 on Θ_{Match} with $M_1(S_{11}) = m_{p_1}(\{\overset{S}{\not\sim}\}) = .60, M_1(S_{12}) = m_{p_1}(\{\overset{S}{\not\cap}\}) = .32, M_1(S_{13}) = m_{p_1}(\{\overset{S}{\not\supset}, \overset{S}{\not\cap}, \overset{S}{\not\cap}\}) = .08$.

Similarly, m_{p_2} and m_{p_3} are translated into BPAs M_2 and M_3 on Θ_{Match}:

#	p_1	p_2	p_3	$p_4 - p_{30}$
Match_{N1}	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$
Match_{N2}	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$	$\overset{s}{\not\supset}$	$\overset{s}{\not\subset}$
Match_{N3}	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$	$\overset{s}{\subset}$	$\overset{s}{\not\subset}$
Match_{N4}	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$	$\overset{s}{\supset}$	$\overset{s}{\not\subset}$
Match_{N5}	$\overset{s}{\not\subset}$	$\overset{s}{\not\supset}$	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$
Match_{N6}	$\overset{s}{\not\subset}$	$\overset{s}{\not\supset}$	$\overset{s}{\not\supset}$	$\overset{s}{\not\subset}$
Match_{N7}	$\overset{s}{\not\subset}$	$\overset{s}{\not\supset}$	$\overset{s}{\subset}$	$\overset{s}{\not\subset}$
Match_{N8}	$\overset{s}{\not\subset}$	$\overset{s}{\not\supset}$	$\overset{s}{\supset}$	$\overset{s}{\not\subset}$
Match_{N9}	$\overset{s}{\not\subset}$	$\overset{s}{\subset}$	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$
Match_{N10}	$\overset{s}{\not\subset}$	$\overset{s}{\subset}$	$\overset{s}{\not\supset}$	$\overset{s}{\not\subset}$
Match_{N11}	$\overset{s}{\not\subset}$	$\overset{s}{\subset}$	$\overset{s}{\subset}$	$\overset{s}{\not\subset}$
Match_{N12}	$\overset{s}{\not\subset}$	$\overset{s}{\subset}$	$\overset{s}{\supset}$	$\overset{s}{\not\subset}$
Match_{N13}	$\overset{s}{\not\subset}$	$\overset{s}{\supset}$	$\overset{s}{\not\subset}$	$\overset{s}{\not\subset}$
Match_{N14}	$\overset{s}{\not\subset}$	$\overset{s}{\supset}$	$\overset{s}{\not\supset}$	$\overset{s}{\not\subset}$
Match_{N15}	$\overset{s}{\not\subset}$	$\overset{s}{\supset}$	$\overset{s}{\subset}$	$\overset{s}{\not\subset}$
Match_{N16}	$\overset{s}{\not\subset}$	$\overset{s}{\supset}$	$\overset{s}{\supset}$	$\overset{s}{\not\subset}$

Table 4.5: First 16 possible schema mappings

$$S_{21} = \{\text{Match}_1, \ldots, \text{Match}_4, \text{Match}_{17}, \ldots \text{Match}_{20}, \text{Match}_{33}, \ldots, \text{Match}_{36}, \text{Match}_{49}, \ldots, \text{Match}_{52}\}$$

$$S_{22} = \{\text{Match}_5, \ldots, \text{Match}_8, \text{Match}_{21}, \ldots \text{Match}_{24}, \text{Match}_{37}, \ldots, \text{Match}_{40}, \text{Match}_{53}, \ldots, \text{Match}_{56}\}$$

$$S_{23} = \{\text{Match}_5, \ldots, \text{Match}_{16}, \text{Match}_{21}, \ldots \text{Match}_{32}, \text{Match}_{37}, \ldots, \text{Match}_{48}, \text{Match}_{53}, \ldots, \text{Match}_{64}\}$$

$M_2(S_{21}) = .20, M_2(S_{22}) = .64, M_2(S_{23}) = .16$

$$S_{31} = \{\text{Match}_1, \text{Match}_5, \text{Match}_9, \text{Match}_{13}, \text{Match}_{17}, \text{Match}_{21}, \text{Match}_{25}, \text{Match}_{29}, \text{Match}_{33}, \text{Match}_{37}, \text{Match}_{41}, \text{Match}_{45}, \text{Match}_{49}, \text{Match}_{53}, \text{Match}_{57}, \text{Match}_{61}\}$$

$$S_{32} = \{\text{Match}_2, \text{Match}_6, \text{Match}_{10}, \text{Match}_{14}, \text{Match}_{18}, \text{Match}_{22}, \text{Match}_{26}, \text{Match}_{30}, \text{Match}_{34}, \text{Match}_{38}, \text{Match}_{42}, \text{Match}_{46}, \text{Match}_{50}, \text{Match}_{54}, \text{Match}_{58}, \text{Match}_{62}\}$$

$$S_{33} = \{\text{Match}_2, \ldots, \text{Match}_4, \text{Match}_6, \ldots \text{Match}_8, \text{Match}_{10}, \ldots, \text{Match}_{12}, \text{Match}_{14}, \ldots, \text{Match}_{16}, \text{Match}_{18}, \ldots \text{Match}_{20}, \text{Match}_{22}, \ldots, \text{Match}_{24}, \text{Match}_{26}, \ldots, \text{Match}_{28}, \text{Match}_{30}, \ldots, \text{Match}_{32}, \text{Match}_{34}, \ldots, \text{Match}_{36}, \text{Match}_{38}, \ldots, \text{Match}_{40}, \text{Match}_{42}, \ldots, \text{Match}_{44}, \text{Match}_{46}, \ldots, \text{Match}_{48}, \text{Match}_{50}, \ldots, \text{Match}_{52}, \text{Match}_{54}, \ldots, \text{Match}_{56}, \text{Match}_{58}, \ldots, \text{Match}_{60}, \text{Match}_{62}, \ldots, \text{Match}_{64}\}$$

$M_3(S_{31}) = .20, M_3(S_{32}) = .64, M_3(S_{33}) = .16$

No mass is assigned on the rest of the schema mappings in Θ_{Match}.

The final BPA M_{Match} is produced by combining the above translated masses M_1, M_2 and M_3:

$$\begin{aligned}
M_{Match}(\{\text{Match}_1\}) &= .024 \\
M_{Match}(\{\text{Match}_2\}) &= .0768 \\
M_{Match}(\{\text{Match}_2, \text{Match}_3, \text{Match}_4\}) &= .0192 \\
M_{Match}(\{\text{Match}_5\}) &= .0768 \\
M_{Match}(\{\text{Match}_6\}) &= .24576 \\
M_{Match}(\{\text{Match}_6, \text{Match}_7, \text{Match}_8\}) &= .06144 \\
M_{Match}(\{\text{Match}_5, \text{Match}_9, \text{Match}_{13}\}) &= .0192 \\
M_{Match}(\{\text{Match}_6, \text{Match}_{10}, \text{Match}_{14}\}) &= .06144 \\
M_{Match}(\{\text{Match}_6, \text{Match}_7, \text{Match}_8, \text{Match}_{10}, \text{Match}_{11}, \\
\text{Match}_{12}, \text{Match}_{14}, \text{Match}_{15}, \text{Match}_{16}\}) &= .01536 \\
M_{Match}(\{\text{Match}_{17}\}) &= .0128 \\
M_{Match}(\{\text{Match}_{18}\}) &= .04096 \\
M_{Match}(\{\text{Match}_{18}, \text{Match}_{19}, \text{Match}_{20}\}) &= .01024
\end{aligned}$$

$$
\begin{aligned}
&M_{Match}(\{\text{Match}_{21}\}) &&= .04096\\
&M_{Match}(\{\text{Match}_{22}\}) &&= .131072\\
&M_{Match}(\{\text{Match}_{22}, \text{Match}_{23}, \text{Match}_{24}\}) &&= .032768\\
&M_{Match}(\{\text{Match}_{21}, \text{Match}_{25}, \text{Match}_{29}\}) &&= .01024\\
&M_{Match}(\{\text{Match}_{22}, \text{Match}_{26}, \text{Match}_{30}\}) &&= .032768\\
&M_{Match}(\{\text{Match}_{22}, \text{Match}_{23}, \text{Match}_{24}, \text{Match}_{26}, \text{Match}_{27},\\
&\text{Match}_{28}, \text{Match}_{30}, \text{Match}_{31}, \text{Match}_{32}\}) &&= .008192\\
&M_{Match}(\{\text{Match}_{17}, \text{Match}_{33}, \text{Match}_{49}\}) &&= .0032\\
&M_{Match}(\{\text{Match}_{18}, \text{Match}_{34}, \text{Match}_{50}\}) &&= .01024\\
&M_{Match}(\{\text{Match}_{18}, \text{Match}_{19}, \text{Match}_{20}, \text{Match}_{34}, \text{Match}_{35},\\
&\text{Match}_{36}, \text{Match}_{50}, \text{Match}_{51}, \text{Match}_{52}\}) &&= .00256\\
&M_{Match}(\{\text{Match}_{21}, \text{Match}_{37}, \text{Match}_{53}\}) &&= .01024\\
&M_{Match}(\{\text{Match}_{22}, \text{Match}_{38}, \text{Match}_{54}\}) &&= .032768\\
&M_{Match}(\{\text{Match}_{22}, \text{Match}_{23}, \text{Match}_{24}, \text{Match}_{38}, \text{Match}_{39},\\
&\text{Match}_{40}, \text{Match}_{54}, \text{Match}_{55}, \text{Match}_{56}\}) &&= .008192\\
&M_{Match}(\{\text{Match}_{21}, \text{Match}_{25}, \text{Match}_{29}, \text{Match}_{37}, \text{Match}_{41},\\
&\text{Match}_{45}, \text{Match}_{53}, \text{Match}_{57}, \text{Match}_{61}\}) &&= .00256\\
&M_{Match}(\{\text{Match}_{22}, \text{Match}_{26}, \text{Match}_{30}, \text{Match}_{38}, \text{Match}_{42},\\
&\text{Match}_{46}, \text{Match}_{54}, \text{Match}_{58}, \text{Match}_{62}\}) &&= .008192\\
&M_{Match}(\{\text{Match}_{22}, \text{Match}_{23}, \text{Match}_{24}, \text{Match}_{26}, \text{Match}_{27},\\
&\text{Match}_{28}, \text{Match}_{30}, \text{Match}_{31}, \text{Match}_{32}, \text{Match}_{38},\\
&\text{Match}_{39}, \text{Match}_{40}, \text{Match}_{42}, \text{Match}_{43}, \text{Match}_{44},\\
&\text{Match}_{46}, \text{Match}_{47}, \text{Match}_{48}, \text{Match}_{54}, \text{Match}_{55},\\
&\text{Match}_{56}, \text{Match}_{58}, \text{Match}_{59}, \text{Match}_{60}, \text{Match}_{62},\\
&\text{Match}_{63}, \text{Match}_{64}\}) &&= .002048
\end{aligned}
$$

◇

Having derived M_{Match} and Θ_{Match}, the belief and/or the plausibility of each possible schema mapping can be computed. However, since the number of possible schema mappings is exponential to the number of pairs N in the matching task, even just computing M_{Match} and Θ_{Match}, is time consuming. In addition, the belief and/or the plausibility of each possible schema mapping is not even necessary. What is useful is the belief and/or the plausibility of the schema mappings that are the most probable ones, *i.e.* the **top-K schema mappings**.

Definition 4.11. Top-K schema mappings A list of K tuples $(\text{Match}_i, l_i), 1 \leq i \leq \text{K}$ each one associating a schema mapping Match_i with a degree of belief l_i is called **top-K schema mappings**. The definition also holds for plausibility. The top-K schema mappings are ordered based on l_i in descending order.

□

Existing matching approaches are equivalent to identifying the top-1 schema mappings.

The problem now lies in identifying the top-K schema mappings without having to identify all 6^N possible schema mappings nor compute their belief or plausibility. A solution to this problem is provided in the next chapter (Section 5.2). The solution is based on the fact that the belief/plausibility of a schema mapping can be identified, without computing M_{Match}, as the product of the beliefs/plausibilities of the semantic relationships the schema mapping is composed of.

Lemma 1. The belief b_l of a schema mapping Match_l which belongs to the list of top-K schema mappings $[(\text{Match}_1, b_1), \ldots, (\text{Match}_K, b_K)]$ is equivalent to the product of the beliefs of the semantic relationships Match_l is composed of:

$$Bel(\{\text{Match}_l\}) = \sum_{A \subseteq \{\text{Match}_l\}, A \in \Theta_{Match}} M_{Match}(A) = \prod_{1 \leq j \leq N} Bel(\text{Match}_l[j])$$

where $\text{Match}_l[j]$ is the semantic relationship selected for pair j in Match_l and $Bel(\text{Match}_l[j])$ is the belief assigned to $\text{Match}_l[j]$.

Proof. Since the set $\{\text{Match}_l\}, 1 \leq l \leq K$ is a singleton event of Θ_{Match} and M_{Match} is a BPA, *i.e.* $M_{Match}(\emptyset) = 0$, the only subset A that satisfies the conditions $A \in \Theta_{Match}, A \subseteq \{\text{Match}_l\}$ is the set $A = \{\text{Match}_l\}$. Therefore, computing the belief of $\{\text{Match}_l\}$ based on Definition 4.8 we have:

$$Bel(\{\text{Match}_l\}) = \sum_{A \subseteq \{\text{Match}_l\}, A \in \Theta_{Match}} M_{Match}(A) = M_{Match}(\{\text{Match}_l\}) \tag{4.4}$$

To compute $M_{Match}(\{\text{Match}_l\})$ we consecutively apply Dempster's rule. For example, $M_{1\oplus 2}(\{\text{Match}_l\}) = M_1 \oplus M_2(\{\text{Match}_l\})$ is computed as follows

$$M_{1\oplus 2}(\{\text{Match}_l\}) = \frac{\sum_{S_{1i} \subseteq \Theta_{Match}, S_{2j} \subseteq \Theta_{Match}, S_{1i} \cap S_{2j} = \{\text{Match}_l\}} M_1(S_{1i}) M_2(S_{2j})}{\sum_{S_{1i} \subseteq \Theta_{Match}, S_{2j} \subseteq \Theta_{Match}, S_{1i} \cap S_{2j} \neq \emptyset} M_1(S_{1i}) M_2(S_{2j})} \tag{4.5}$$

where M_1 is the BPA derived from the BPA m_{p_1} on the semantic relationships of pair p_1, and S_{1i} is the set of schema mappings corresponding to the sets of relationships $rels_{1i}$ the BPA m_{p_i} assigns probability mass to. Similarly for M_2.

Based on Equation (4.3), the following hold:

$$\begin{aligned} &\forall \text{Match}_q \in S_{1i} : \text{Match}_q[1] \in rels_{1i}, \\ &\forall \text{Match}_q \in S_{2j} : \text{Match}_q[2] \in rels_{2j}. \end{aligned} \tag{4.6}$$

Since the schema mappings in Θ_{Match} are produced by computing all possible combinations of semantic relationships amongst the pairs, there must be at least one $\text{Match}_q \in S_{1i} \cap S_{2j}$ such that $\text{Match}_q[1] \in rels_{1i}$ and $\text{Match}_l[2] \in rels_{2j}$. Thus,

$\forall i, j : S_{1i} \cap S_{2j} \neq \emptyset$, and therefore the following is true

$$\sum_{S_{1i} \subseteq \Theta_{Match}, S_{2j} \subseteq \Theta_{Match}, S_{1i} \cap S_{2j} \neq \emptyset} M_1(S_{1i}) M_2(S_{2j}) = 1$$

and Equation (4.5) becomes

$$M_{1 \oplus 2}(\{\text{Match}_l\}) = \sum_{S_{1i} \subseteq \Theta_{Match}, S_{2j} \subseteq \Theta_{Match}, S_{1i} \cap S_{2j} = \{\text{Match}_l\}} M_1(S_{1i}) M_2(S_{2j}) \qquad (4.7)$$

In order for $S_{1i} \cap S_{2j} = \{\text{Match}_l\}$, $rels_{1i}$ and $rels_{2j}$ must include $\text{Match}_l[1]$ and $\text{Match}_l[2]$ respectively (Equation (4.6)), but they cannot include any other relationships. If either $rels_{1i}$ or $rels_{2j}$ included more relationships, then the intersection $S_{1i} \cap S_{2j}$ would contain more than one schema mappings and not only Match_l. For example, if $\text{Match}_l[1] = \overset{\text{S}}{=}$ and $\text{Match}_l[2] = \overset{\text{S}}{\subset}$, while $rels_{1i} = \{\overset{\text{S}}{=}, \overset{\text{S}}{\not\cap}\}$ and $rels_{2j} = \{\overset{\text{S}}{\subset}\}$, then the intersection $S_{1i} \cap S_{2j}$ would contain not only schema mapping Match_l, but also a schema mapping Match_v, with $\text{Match}_v[1] = \overset{\text{S}}{\not\cap}$ and $\text{Match}_v[2] = \overset{\text{S}}{\subset}$. Thus, we must have that $rels_{1i} = \{\text{Match}_l[1]\}$ and $rels_{2j} = \{\text{Match}_l[2]\}$.

In addition, we have that there is only one pair S_{1i} and S_{2j} that satisfy the condition $S_{1i} \cap S_{2j} = \{\text{Match}_l\}$. For example, if there were S'_{1i} and S'_{2j} such that $S'_{1i} \cap S'_{2j} = \{\text{Match}_l\}$, then as explained previously these S'_{1i} and S'_{2j} can only correspond to relationships $rels_{1i} = \{\text{Match}_l[1]\}$ and $rels_{2j} = \{\text{Match}_l[2]\}$. But then $S'_{1i} = S_{1i}$ and $S'_{2j} = S_{2j}$.

Therefore, from Equation (4.7) we have that

$$M_{1 \oplus 2}(\{\text{Match}_l\}) = M_1(S_{1i}) M_2(S_{2j}), \text{ where } S_{1i} \cap S_{2j} = \{\text{Match}_l\}$$

Since $rels_{1i} = \{\text{Match}_l[1]\}$ and $rels_{2j} = \{\text{Match}_l[2]\}$ are singleton events of Θ_{rel}, we

have that $m_1(\{\text{Match}_l[1]\}) = Bel(\{\text{Match}_l[1]\})$ and $m_2(\{\text{Match}_l[2]\}) = Bel(\{\text{Match}_l[2]\})$. Based also on the definition of M_1 and M_2, the previous formula becomes:

$$\begin{aligned} M_{1\oplus 2}(\{\text{Match}_l\}) &= m_1(\{\text{Match}_l[1]\})m_2(\{\text{Match}_l[2]\}) \\ &= Bel(\text{Match}_l[1])Bel(\text{Match}_l[2]) \end{aligned}$$

Similarly, we can show by induction that

$$M_{Match}(\{\text{Match}_l\}) = Bel(\{\text{Match}_l[1]\}) \dots Bel(\{\text{Match}_l[N]\}).$$

Thus, Equation (4.4) becomes:

$$\begin{aligned} Bel(\{\text{Match}_l\}) &= Bel(\{\text{Match}_l[1]\}) \dots Bel(\{\text{Match}_l[N]\}) \\ &= \textstyle\prod_{1\leq j\leq N} Bel(\text{Match}_l[j]). \end{aligned}$$

□

Example 4.8. Top-2 Schema Mappings Following on from the previous example, instead of computing the BPA for the 64 possible schema mappings, we just discover the top-2 schema mappings. Depending on user selection, the top-K algorithm can be run on either belief or plausibility.

Based on Table 4.4 and having selected belief to run the top-K algorithm on, in the first iteration of the top-K algorithm will select for each pair the relationship with the highest belief producing schema mapping $\text{Match}_{K=1}$, which includes the following semantic mappings:

(i) $\langle \langle\!\langle \mathsf{paper} \rangle\!\rangle, \overset{\text{S}}{\gamma}, \langle\!\langle \mathsf{book} \rangle\!\rangle \rangle$

(ii) $\langle \langle\!\langle \mathsf{paper}, \mathsf{title} \rangle\!\rangle, \overset{\text{S}}{\eta}, \langle\!\langle \mathsf{book}, \mathsf{title} \rangle\!\rangle \rangle$

(iii) $\langle \langle\!\langle \mathsf{paper}, \mathsf{year} \rangle\!\rangle, \overset{\text{S}}{\eta}, \langle\!\langle \mathsf{book}, \mathsf{year} \rangle\!\rangle \rangle$

The belief on this schema is computed as the product of the beliefs on the relationships selected for each pair (see Table 4.4), *i.e.* $0.6 \times 0.64 \times$

$0.64 = 0.24576$. However, we know that this is actually an incorrect schema mapping. The correct one is selected in the next iteration of the top-K algorithm.

In the second iteration, the top-K algorithm will change the relationship for one of the pairs. The next highest belief comes from the disjointness relationship for pair p_1. Therefore, the schema mapping $\text{Match}_{K=2}$ produced in the second iteration includes the semantic mappings:

(i) $\langle \langle\langle \mathsf{paper} \rangle\rangle, \overset{\text{S}}{\cap}, \langle\langle \mathsf{book} \rangle\rangle \rangle$

(ii) $\langle \langle\langle \mathsf{paper}, \mathsf{title} \rangle\rangle, \overset{\text{S}}{\cap}, \langle\langle \mathsf{book}, \mathsf{title} \rangle\rangle \rangle$

(iii) $\langle \langle\langle \mathsf{paper}, \mathsf{year} \rangle\rangle, \overset{\text{S}}{\cap}, \langle\langle \mathsf{book}, \mathsf{year} \rangle\rangle \rangle$

The belief of this schema mapping is 0.131072.

Thus, the top-2 schema mappings are: $[(\text{Match}_{K=1}, 0.24576), (\text{Match}_{K=2}, 0.131072)]$.

◇

The previous example (Example 4.8) shows a case where the top-1 schema mapping is not entirely correct. In particular, the semantic mapping for the paper and book objects is wrong. However, by looking at the next most probable schema mapping, we identify the correct schema mapping. Thus, the reason for identifying the top-K schema mappings rather than the top-1 schema mapping in our schema integration framework, is that the top-K schema mappings can potentially provide an improved mapping compared to the top-1 schema mapping.

In the next chapter, in Section 5.2, we explain two approaches for deriving the top-K schema mappings: the exhaustive and the truncated top-K approaches. In the exhaustive top-K approach, two consecutive mappings from the top-K schema mappings may differ only in the semantic mapping of a few pair of objects. Thus, schema mapping at rank $i + 1$ in the exhaustive top-K approach might not offer a significant improvement over the schema mapping at rank i. However, the exhaustive

top-K approach can guarantee improved schema mappings in certain situations (see Section 5.3.4). In the truncated top-K approach, two consecutive schema mappings have different semantic mappings for groups of pairs of objects, which means that the two schema mappings differ significantly. Thus, schema mapping at rank $i+1$ in the truncated top-K approach could potentially be a significant improvement over schema mapping at rank i. The drawback of the truncated top-K approach is that some schema mappings, potentially including the schema mapping with the correct semantic mappings for all pairs of objects, are never considered.

4.7.1 Top-1 vs Top-K

In this section we are attempting to identify the cases where either top-1 or top-K matching is preferable.

The purpose of schema matching as explained previously and defined in [90] is the identification of correspondences, or **mappings**, between input schema objects. Based on Section 4.2.2, schema matching is the process that derives a schema mapping between the input schemas. Thus, an obvious objective O1 for the schema matching process is: to identify the single correct schema mapping according to a single expert user's perspective. To satisfy objective O1, it is sufficient to perform a top-1 matching; the expert user can verify the correctness of the single schema mapping produced or correct it.

As we have previously explained and illustrated in Example 4.8, the most probable schema mapping (top-1) is not always the correct schema mapping. Thus, an obvious extension to the above objective for schema matching is objective O2: to identify a single schema mapping that the expert user is satisfied with. We assume that expert satisfaction increases monotonically with the number of correct (according to the expert user) semantic mappings identified. To satisfy objective O2, a top-K approach is more preferable since the expert will have K schema mappings to choose

from and select the one she is more satisfied with.

Notice here that we are only interested in low-level objectives regarding the schema mapping produced from the matching process. High-level objectives can also be defined for matching, *e.g.* how the schema mapping discovered during matching can affect the cover of correct answers returned when the final schema is queried, but these are out of the scope of our research.

On a fully automatic schema matching task, an expert user is not available to confirm neither the correctness of the schema mapping produced nor her satisfaction with the mapping. Thus, the above two objectives are not applicable in this case. In a fully automated setting, the schema mapping produced by the matching process has to be used directly in the schema integration process. If top-1 matching is applied, then a single integrated schema is produced, thus a single set of answers is returned for each query. If the top-K approach is used, multiple sets of answers will be returned for each query, thus potentially increasing the number of correct answers covered, providing more flexibility and accuracy [37].

Regarding the manual cost of matching, in a fully automated setting there is no manual cost by definition. Regarding objectives O1 and O2, the correction of a schema mapping or the selection of a schema mapping out of the top-K requires user interaction and a user cost which we are now going to estimate.

Note that regarding the correction of a schema mapping we cannot use approaches that assume data mappings as part of their schema mappings. For example, in [25], schema mappings consist of *tuple generating dependencies* which is a data mapping formalism. These approaches require the existence of schemas that can be queried, and thus they are more relevant in data exchange scenarios where data mappings are defined between pre-existing source and target schemas. In our approach, we deal with schema integration where the integrated schema does not exist and cannot be queried. In our approach, the schema mapping (Definition 4.6) consists of semantic

mappings which have been defined between the input schemas and not between the input and the integrated schemas.

We consider the process of correcting a schema mapping as follows: for each pair p_i of objects in the schema mapping, the user has to: (a) identify the correct semantic relationship for p_i, *i.e.* select one of the possible relationships for p_i, (b) check whether the schema mapping correctly identifies the semantic relationship of p_i, and if not, (c) alter the semantic mapping of p_i with the correct relationship. Assume that the manual cost of identifying the correct relationship for any pair of objects, *i.e.* step (a), is constant and equal to $identify_{rel}$. The costs of steps (b) and (c) are also constant for any pair of objects and aggregated together they are equal to fix_{pair}. We have that $fix_{pair} << identify_{rel}$, since the identification of the correct semantic relationship of a pair of objects is a highly intelligent process, Thus, the cost $Cost_{fix}$ of correcting a schema mapping of N pairs can be specified as

$$Cost_{fix} = identify_{rel} \times N \tag{4.8}$$

which is proportional to the number N of pairs in the matching task.

Note that the above equation estimates the complexity of the process of correcting a schema mapping. For example, it does not specify the mapping selection process which should be followed to identify the semantic relationship for each pair of objects, *i.e.* step (a), which the user could perform either manually or with the help of a tool, such as Clio [76] and the approach in [106].

Regarding the complexity of selecting a schema mapping out of the top-K, assume that K=2 and that we compare schema mappings sm_1 and sm_2 which have N pairs of objects. We define the cost of this comparison, $Cost_{pick}(sm_1, sm_2)$, as

$$Cost_{pick}(sm_1, sm_2) = \Sigma_{1 \leq i \leq N} pick_{rel}(p_i)$$

where $pick_{rel}(p_i)$ is the cost of comparing the semantic relationships pair p_i is assigned in sm_1 and sm_2, and selecting one. Note that again we are not interested in the details of how this selection is performed. If p_i has been assigned the same relationship in both sm_1 and sm_2, then $pick_{rel}(p_i) = 0$. Thus, the complexity of comparing two schema mappings is proportional to the number of pairs whose relationship differs in the two mappings.

Based on the above cost function, the cost of comparing K schema mappings $Cost_{pick}($ $sm_1, \ldots, sm_k)$ is the sum of costs of comparing the K schema mappings pairwise. Since K is a constant, the complexity of $Cost_{pick}(sm_1, \ldots, sm_k)$ is also proportional to the number of pairs in the matching task whose relationship differs in the K mappings.

In terms of the complexity of top-1 and top-K matching, the first question is whether the identification of a USM is more complex than the identification of a single certain semantic mapping. As it will be shown in Chapter 5, the implementation of a matching expert that identifies USMs rather than certain semantic mappings requires more space. If s is the space that is required for deriving certain semantic mappings, then $2^6 \times s$ is the space required for deriving USMs, where 2^6 is the number of all possible sets of semantic relationships, since we use 6 semantic relationships in total. Thus, the space complexity of deriving USMs is of the same order as for certain semantic mappings. In addition, the combination of USMs is also of the same order as the combination, *e.g.* using a weighted average function, of similarity degrees most existing approaches adopt, *e.g.* [35, 65, 33, 46].

The main complexity issue caused by the introduction of uncertainty in the result of the schema matching process is the exponential number 6^N of schema mappings that need to be managed. While in the top-1 approach only one schema mapping is derived, when uncertainty is exposed the belief/plausibility of each possible schema mapping needs to be derived, *i.e.* exponential space 6^N is required. However, by just restricting the number of schema mappings required and only focusing on the

top-K schema mappings, the space complexity of the process is just $K \times s$ the space complexity s of existing approaches.

4.8 Top-K Schema Merging

Based on the top-K schema mappings identified during matching, each one annotated with a belief or plausibility degree, K schema merging tasks are defined.

Each merge task corresponds to one of the top-K schema mappings and produces an integrated schema. Each integrated schema will be annotated with the belief or plausibility degree of the schema mapping it was derived from. Due to the fact that our schema merging process (introduced in Chapter 6) may produce identical schemas for multiple distinct schema mappings, the top-K schema mappings result in at most K integrated schemas, which can be ordered based on their associated belief or plausibility. For example, if two schema mappings Match_{Ni} and Match_{Nj}, $i \neq j$, with associated belief or plausibility l_i and l_j respectively, map to identical integrated schema S_{ij}, then the belief or plausibility of S_{ij} is equal to $l_i + l_j$ normalized.

Definition 4.12. Top-K integrated schemas From top-K schema mappings $[(\text{Match}_{N1}, l_1), \ldots, (\text{Match}_{NK}, l_K)]$ a ranked list S of at most K integrated schemas is derived. Each integrated schema $S_i \in S, 1 \leq i \leq \text{K}$, corresponds to a set of schema mappings $\{\text{Match}_{Nm_1^i}, \ldots, \text{Match}_{Nm_n^i}\}, 1 \leq m_j^i \leq \text{K}$, and is annotated with the sum of the belief or plausibility degrees of schema mappings $\text{Match}_{Nm_j^i}$, *i.e.* $l_{m_1^i} + \ldots + l_{m_n^i}$, normalized. Thus, we have

$$S = [(S_1, \frac{l_{m_1^1} + \ldots + l_{m_n^1}}{l_1 + \ldots + l_K}), \ldots, (S_j, \frac{l_{m_1^j} + \ldots + l_{m_n^j}}{l_1 + \ldots + l_K})], 1 \leq j \leq \text{K}.$$

S is called the **top-K integrated schemas**.

□

4.9 Summary

In this chapter, we have presented our framework for the representation of uncertainty in schema matching and schema merging.

We show that initially in the schema matching process, the semantics of the schema objects need to be identified, *i.e.* the real world entities the objects represent. Based on these semantics, the semantic mappings can be identified. However, the semantics of the objects are not available to an automatic matching tool, therefore the mappings it produces are highly uncertain. In our framework, we aim to represent this schema matching uncertainty, and we specify four features that this representation should support: (a) ability to rank semantic relationships for each pair of objects, (b) assignment of a degree of uncertainty to each singleton semantic relationship for each pair of objects, (c) ability to express ignorance, and (d) ability to express certainty. In our framework, we have adopted Dempster-Shafer's theory to represent schema matching uncertainty.

In this chapter, we first introduce the notion of the uncertain semantic mapping, which assigns probability masses to sets of semantic relationships for a pair of objects. These probability masses represent the certainty of the tool about the correctness of a semantic relationship for a specific pair of objects. Thus, for each pair multiple semantic relationships are possible, each one with a degree of certainty. Then, we show how uncertain semantic mappings, based on Dempster-Shafer's theory, support the four features required. In Section 2.2.3, where we reviewed the two existing approaches [46, 81] that can be used to represent schema matching uncertainty on multiple semantic relationships, we saw that [46] is based on fuzzy set theory and [81] on probabilities. In this chapter (Section 4.6), we explain why our approach is more suitable than these approaches to model uncertainty. In particular, we show that Dempster-Shafer's theory can represent probabilities and that fuzzy set theory does not fully support the features required.

The final result of schema matching in our framework is an uncertain schema mapping, which is produced by combining uncertain semantic mappings. An uncertain schema mapping assigns probability masses to sets of schema mappings; it essentially assigns probability masses to sets of possible integrated schemas. Because the number of possible schema mappings is exponential to the number of pairs of objects in the matching task, the identification of the level of uncertainty of each schema mapping, and thus each integrated schema, is of exponential space complexity. Therefore in our framework, we introduce the notion of the top-K schema mappings and top-K integrated schemas, which allow the identification of the K mappings and the at most K schemas the tool is most certain about. In this chapter, we have also explained the complexity of selecting a schema mapping out of the top-K.

Chapter 5

Top-K Schema Matching

In the previous chapter the framework of our schema integration approach was presented. Our approach improves upon existing methodologies by extending the notion of mappings between schema objects. In particular, in our framework the uncertainty of the correct semantic relationship between each pair of schema objects is explicitly represented in **uncertain semantic mapping**s (USMs), which give rise to multiple possible schema mappings for each integration task, which we call **top-K schema mappings**, and multiple possible integrated schemas, which we call **top-K integrated schemas**. Our approach subsumes existing matching and merging approaches that do not externalize schema matching uncertainty on multiple semantic mappings and whose outcome is a single integrated schema.

In our proposed architecture (Section 4.1), the USMs of a matching task are discovered by the Match component. The Top-K component translates these USMs into top-K schema mappings. In this chapter we present our implementation of the Match and Top-K components. In our implementation of the Match component we have adapted existing schema matching, ontology aligning, string matching and word similarity research software, as well as introduced new simple matching algorithms.

Additionally, in this chapter a thorough experimental evaluation of our matching implementation is presented. The evaluation is based on a well-known set of user, schema and schema mapping data previously used for the evaluation of existing matching software [73]. The experimental results show that:

- our implementation can improve the accuracy of the matching algorithms we experimented with for deriving a single schema mapping,
- the top-K schema mappings produced in our framework can improve the accuracy of our top-1 matching.

The structure of this chapter is as follows. In Section 5.1, we present the algorithms used in the Match component to compare schema objects. These algorithms produce a *similarity degree* for each pair of objects just like existing matching tools. In the same section, we show how these similarity degrees can be translated into USMs. In Section 5.2, we present our implementation of the Top-K component. We present and compare two possible approaches: the exhaustive and the truncated top-K approaches. In Section 5.3, we present our test bed for the experimental evaluation, we explain the metric used to evaluate the matching process, and finally we show our experimental results for top-1 and top-3 schema matching.

5.1 The Match Component

As illustrated in Figure 4.1, the Match component takes as input a list of schema object pairs $p_j, 1 \leq j \leq M$ and produces an uncertain schema mapping, *i.e.* M USMs (Definition 4.10).

The Match component consists of a set of N experts $E_i, 1 \leq i \leq N$, which produce USMs, and an aggregator which aggregates these USMs. The task of each expert E_i is to independently compare each pair $p_j, 1 \leq j \leq M$, of objects and produce

a USM $usm_{i,j}$ for each pair. The Aggregator combines the USMs produced by the experts for each pair of objects using Dempster's rule and produces a single final USM usm_j for each pair. For example, for pair p_1 the USMs $usm_{1,1}, \ldots, usm_{N,1}$ produced by the N experts are aggregated to give a final USM usm_1. Thus, the aggregator produces a list of USMs $usm_j, 1 \leq j \leq M$, one USM for each pair of objects. This list constitutes an uncertain schema mapping, which is the output of the Match component.

5.1.1 Overview of the Match Experts

In our implementation of the Match component, we have included new experts and have adapted experts from existing schema matching, ontology aligning, string matching and word similarity research software.

The main task of each expert E_i is to compare each pair of objects p_j and based on that comparison produce a USM $usm_{i,j}$ for the pair. We remind the reader that a USM on pair p_j assigns probability masses to the possible semantic relationships for the pair, *i.e.* it assigns probability masses to subsets of $\Theta_{rel} = \{\overset{S}{=}, \overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}$. Thus the main task of each expert is to compare each pair of objects and based on that comparison to identify the probabilities of the semantic relationships for that pair.

In existing schema matching approaches, the result of the comparison of a pair of objects is a normalized value in the [0,1] range, which is usually called **similarity degree** [90]. The implicit assumption in these approaches is that the higher the similarity degree is, the more likely is that the pair of objects are compatible. Analogously, the lower the similarity degree is, the more likely is that the pair of objects are incompatible. Thus, existing approaches assume *monotonicity* [46] between the similarity degree and the level of compatibility. However, there is no existing matching approach that is shown to be monotonic [45].

In our implementation, we do not make any assumptions about the monotonicity of the similarity degrees. Each Match expert produces a USM for each pair of objects based on the similarity degree the expert has computed by comparing the two objects. The correlation between the similarity degree and the semantic relationship this degree implies is left to the expert to identify.

In order to produce a USM for each pair p_j, each expert needs to identify the probabilities of the semantic relationships for p_j, based on the similarity degree it has produced for p_j. To this end, each expert stores statistical data from previous matching tasks that associate the similarity degrees of pairs and their user-confirmed semantic relationships. For example, based on previous matching tasks an expert could have stored the statistical information that 5 pairs of objects that the user has confirmed to be compatible have similarity degrees in the [0.9,1] range, while only one pair, which is incompatible according to the user, has similarity degree in the same [0.9,1] range. Based on this information, any new pair of objects with a similarity degree in the [0.9,1] range will be assigned a $\frac{5}{6}$ probability on compatibility and a $\frac{1}{6}$ probability on incompatibility. Such probabilities can be used for the definition of the USM for each pair of objects.

We say that each result of the Match component is *user-centred* since the statistical data held by each expert, and therefore the probabilities and USMs that the expert produces, depend on the user's feedback from previous matching tasks, *e.g.* the result in the example above would be different if another user for the same tasks had identified 4 compatible and 2 incompatible pairs that have similarity values in the [0.9,1] range. According to this user, any new pair of objects with a similarity degree in the [0.9,1] range will be assigned a $\frac{4}{6}$ probability on compatibility and a $\frac{2}{6}$ probability on incompatibility.

5.1.2 The matching algorithms

The following matching algorithms have been implemented in each expert. They do not constitute a complete list of all possible schema matching algorithms, but they cover the main two matching approaches, schema-level and instance-level matching [90]. Schema-level matching covers both name comparison and structural comparison. We allow for both linguistic name comparison, with the help of the WordNet taxonomy, and syntactic name comparison, using a string matching algorithm. For structural comparison, we have adopted the Similarity Flooding algorithm [73]. Instance-level matching is useful to exclude possible semantic relationships. Each algorithm produces a similarity degree, *i.e.* a value in the [0,1] range, for each pair of schema objects.

- WordNet: The WordNet expert uses the WordNet taxonomy [42] to linguistically compare schema objects. WordNet is a lexical taxonomy with semantic associations between words, such as kind-of, part-of, and synonym-of associations. The expert, for each pair of schema objects, extracts the names of the objects, treats these names as words in WordNet and compares them by applying the Lin algorithm [63]. The implemented expert reuses the RESuLT [104] implementation of the Lin algorithm. In brief, the algorithm defines as similarity between two words W_1 and W_2 the ratio between the amount of information needed to state how common W_1 and W_2 are and the amount of information needed to describe W_1 and W_2. The commonality of any two concepts W_1 and W_2 is identified based on the first common parent word W_p of W_1 and W_2 in the WordNet taxonomy.

  In more detail, the algorithm is based on information theory and computes the amount of information needed to describe a word W as the negative logarithm of the probability of W, $-\log P(W)$. The probability $P(W)$ is provided by WordNet. WordNet defines a probability distribution over the words in its

taxonomy based on each word's frequency [42].

The information needed to describe W_1 and W_2 can now be computed as $-\log P(W_1) - logP(W_2)$ and the commonality of the two words based on the first common parent word W_p is $-\log(P(W_p))$. The similarity is given by the ratio:

$$sim(W_1, W_2) = \frac{2 \times -\log(P(W_p))}{-\log P(W_1) - logP(W_2)}$$

Based on the above formula, for any two identical, or synonymous, words the similarity degree is 1. In case the name of any of the two schema objects is not part of WordNet, the expert has been adapted to produce similarity degree `NaN`, *i.e.* not a number. Later on in this section, we will see how these similarity degrees are handled.

- Rondo String Matching: The Rondo String Matching expert uses a string matching algorithm to syntactically compare the names of schema objects. For this expert we have reused the implementation of previous schema matching approaches [73, 75]. In brief, the algorithm treats the names of the schema objects as plain strings. For each pair of objects, it splits the name of each object into a set of words. Then it compares the words in the two sets, looking for common prefixes and suffixes. Additionally, it uses term frequencies to reduce the impact of common terms in large schemas.

  In more detail, the names n_1, n_2 of each pair of objects s_1, s_2 are split into sets of words sw_1 and sw_2 respectively. Non-letter characters as well as upper-case characters are used to identify the beginning of words. Same-case letters are grouped together forming words, as well as an upper case letter followed by a group of lower-case letters. For example, the string `LCDScreenSize` is split into the set {`LCD`,`Screen`,`Size`}, and the string `LCDMonitor` is split into {`LCD`,`Monitor`}.

For each pair of words $w_i \in sw_1$ and $w_j \in sw_2$, if the two words are identical then the similarity is $word\_similarity(w_i, w_j) = 1$. The words are also compared looking for common prefixes and suffixes. If *pos* is the position of the first letter mismatch, *i.e.* a prefix and/or a suffix has been detected, then the similarity is $word\_similarity(w_i, w_j) = penalty.pos/length(w_i).pos/length(w_j)$, where the default value of *penalty* is 0.5 and the function $length(w)$ returns the length of word w. Note that since for each pair of words there is a prefix and suffix word similarity, the maximum of the two is selected.

Finally, the similarity degree returned by this algorithm for objects s_1, s_2 with names n_1, n_2 respectively is:

$$sim(n_1, n_2, S_1, S_2) = \sum_{i,j} \frac{max(word\_similarity(w_i, w_j))}{freq(w_i, S_1).freq(w_j, S_2)}.$$

where $freq(w, S)$ returns the number of times the word w appears inside the sets of words created for the objects of schema S.

- Similarity Flooding: The Similarity Flooding expert performs a structural comparison based on a graph matching algorithm [73]. The expert is named after this algorithm. In brief, the Similarity Flooding algorithm treats data source schemas as directed labeled graphs and performs an iterative fixpoint computation that propagates the similarities of the nodes to and from their neighbours. The computation stops when between two iterations the similarity of all pairs is not altered significantly. The algorithm is based on the assumption that two nodes are similar if their adjacent nodes in their respective graphs are similar. The expert reuses the implementation provided by the authors of the algorithm [73].

  In more detail, the expert uses the input schema object pairs $[s_i, s_j]$ to built two graph structures of nodes and edges that the Similarity Flooding algorithm requires as input. One graph structure GS_1 corresponds to all the schema

objects s_i in the left-hand side of the input pairs, and the other graph structure GS_2 corresponds to all the schema objects s_j in the right-hand side of the input pairs. Based on the generic classification of constructs presented in Section 3.3, we can translate a schema object of any data model into the required graph structure of the Similarity Flooding algorithm as follows. Each nodal schema object $\langle\langle s \rangle\rangle$ is translated into a node s in the graph. Each link schema objects $\langle\langle l, s_1, s_2 \rangle\rangle$ is translated into a directed edge from node s_1 to node s_2. Each link-nodal schema object $\langle\langle s_1, s_2 \rangle\rangle$ is translated into a node s in the graph and an edge from s_1 to s_2.

After the translation process, the expert feeds the two graphs GS_1 and GS_2 into the Similarity Flooding algorithm. In our implementation, the configuration of the algorithm used is the one that according to [73] produces the *best overall results* and has the *best convergence*. For the sake of completeness this configuration is: the inverse-average formula is used to propagate similarities between adjacent nodes, the fixpoint formula $\sigma^{n+1} = normalize(\sigma^0 + \sigma^n + f(\sigma^0 + \sigma^n))$ is applied to each iteration of the algorithm, the default value 0.05 is used to examine whether the computation has reached the fixed point, and initially all nodes are considered equally similar.

- Cardinality: The Cardinality expert compares the cardinalities of the schema objects, *i.e.* the number of their instances. The similarity degree produced for each pair of objects s_1, s_2 with cardinalities $Card(s_1)$ and $Card(s_2)$ respectively, is:

$$sim(s_1, s_2) = \begin{cases} 1 & Card(s_1) = Card(s_2) \\ 0 & otherwise \end{cases}$$

Thus, for schema objects with equivalent number of instances the algorithm produces similarity degree 1. Since, equivalent ($\stackrel{s}{=}$) schema objects should have equivalent cardinalities, the expert is useful to exclude equivalence mappings.

- Instances-Intersection: The Instances-Intersection expert compares the actual instances of the schema objects. The similarity degree produced by the Instances-Intersection expert for each pair of objects s_1 and s_2 with sample instances $Sample(s_1)$ and $Sample(s_2)$ respectively, is:

$$sim(s_1, s_2) = \frac{2 \times |Sample(s_1) \cap Sample(s_2)|}{|Sample(s_1)| + |Sample(s_2)|}$$

  where $Sample(s_1) \cap Sample(s_2)$ returns a set with the common instances in the two samples, and $|Set|$ returns the size of Set. In our implementation, the sample instances are selected randomly from each object's extent. Instance matching, *i.e.* identifying that two instances are identical, has been implemented based on case-insensitive string equivalence.

  Based on the above formula, the similarity degree is 0 if there are no common instances in the samples of the schema objects. In this case, disjointness and incompatibility might be more probable. If $sim(s_1, s_2) = 1$, *i.e.* $|Sample(s_1)| = |Sample(s_2)| = |Sample(s_1) \cap Sample(s_2)|$, then the objects might be equivalent. In all other cases, the similarity degree is $0 < sim(s_1, s_2) < 1$, which might allow the exclusion of disjointness and incompatibility.

- Instances-Subset: The similarity degree produced by the Instances-Subset expert for each pair of objects s_1 and s_2 with sample instances $Sample(s_1)$ and $Sample(s_2)$ respectively, is:

$$sim(s_1, s_2) = \frac{|Sample(s_1) \cap Sample(s_2)|}{|Sample(s_1)|}$$

  If $sim(s_1, s_2) = 1$ then the object might be equivalent. If $sim(s_1, s_2) < 1$ then the objects might be intersecting.

- Instances-Superset: The similarity degree produced by the Instances-Superset expert for each pair of objects s_1 and s_2 with sample instances $Sample(s_1)$

and $Sample(s_2)$ respectively, is:

$$sim(s_1, s_2) = \frac{|Sample(s_1) \cap Sample(s_2)|}{|Sample(s_2)|}$$

5.1.3 Producing USMs

The aforementioned algorithms produce a similarity degree for each pair of objects. Each expert that implements one of the above algorithms needs to translate each similarity degree into a USM, *i.e.* each expert needs to identify for each pair of objects probability masses for subsets of $\Theta_{rel} = \{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}$, based on the similarity degree it has produced for the pair. In this section, we first describe the methodology we propose to identify these probability masses and then illustrate our implementation.

To identify the probability masses for the USMs, each expert stores statistical data from user-confirmed schema mappings. These mappings could come from previous matching tasks that the user has validated (see Figure 4.1) and they form the training data set of the expert.

Each expert E_i specifies sets of semantic relationships, $rel_{i,j} \subseteq \Theta_{rel}$, and ranges $range_{i,k}$ of similarity degrees, $range_{i,k} = [x, y], 0 \leq x < y \leq 1$. Based on these, if sim is the similarity degree for pair p produced by E_i and rel the semantic relationship for p confirmed by the user, E_i stores statistical data that associate sim with $range_{i,k}$ when $sim \in range_{i,k}$, and associate sim with the sets $rel_{i,j}$ such that $rel \in rel_{i,j}$.

Based on the statistical data, the expert E_i is able to identify the following probabilities:

- $P(range_{i,k})$: the probability that the similarity degree sim' for a pair p' of schema objects falls in the $range_{i,k}$, *i.e.* $sim' \in range_{i,k}$.

	$\{\overset{\text{S}}{\not\sim}\}$	$\overset{\text{S}}{\not\sim}$
$[0, 0.1]$	282	4
$(0.1, 0.2]$	13	2
$(0.2, 0.3]$	18	5
$(0.3, 0.4]$	2	1
$(0.4, 0.5]$	0	0
$(0.5, 0.6]$	0	1
$(0.6, 0.7]$	0	0
$(0.7, 0.8]$	0	1
$(0.8, 0.9]$	0	0
$(0.9, 1.0]$	3	7

Table 5.1: $\langle 0.1, \langle \overset{\text{S}}{\not\sim}, \{\overset{\text{S}}{\not\sim}\}\rangle\rangle$ training table

- $P(rel_{i,j} \wedge range_{i,k})$: the *joint* probability that the user-confirmed relationship rel' for a pair p' belongs to the set $rel_{i,j}$ and that the similarity degree sim' for the pair falls in the $range_{i,k}$, *i.e.* that both $rel' \in rel_{i,j}$ and $sim' \in range_{i,k}$ hold.

Based on these probabilities, the expert can compute $P(rel_{i,j}|range_{i,k})$,

$$P(rel_{i,j}|range_{i,k}) = \frac{P(rel_{i,j} \wedge range_{i,k})}{P(range_{i,k})} \tag{5.1}$$

the *conditional* probability that the relationship for a pair p' of objects belongs in the set $rel_{i,j}$ given that the similarity degree sim' for the pair falls in the $range_{i,k}$.

The implementation of the above methodology is as follows. Each expert includes a *trainer* component. The trainer learns the probabilities, $P(range_{i,k})$ and $P(rel_{i,j} \wedge range_{i,k})$, and can calculate the conditional probabilities $P(rel_{i,j}|range_{i,k})$ based on Equation 5.1. The probability $P(range_{i,k})$ is estimated as the proportion of the training pairs of objects that have a similarity degree in $range_{i,k}$ and the probability $P(rel_{i,j} \wedge range_{i,k})$ is estimated as the proportion of pairs of objects that have have a semantic relationship in $rel_{i,j}$ and the similarity degree of the pair is in the range $range_{i,k}$.

The implementation of each trainer component is based on a two-dimensional *training* table T. An example of a training table is illustrated in Table 5.1. The vertical dimension of the table splits the [0,1] range into smaller intervals, $range_{i,j}$. We remind the reader that the [0,1] range is the range of the similarity degrees produced by the matching algorithms. The horizontal dimension of the training table is distinct non-intersecting subsets $rel_{i,j}$ of the set $\Theta_{rel} = \{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}$. Note that $\bigcup_j rel_{i,j} = \Theta_{rel}$. In Table 5.1, the horizontal dimension is split into two sets: $\{\overset{S}{\not\sim}\}$ and set $\overset{S}{\approx} \equiv \{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}\}$ (Definition 4.5).

Each cell $T[range_{i,k}, rel_{i,j}]$ of table T specifies the number of training pairs p' that have a similarity degree sim' that falls in $range_{i,k}$ and their relationship rel' is a member of $rel_{i,j}$. The sum of all cells $\Sigma_{k,j}T[range_{i,k}, rel_{i,j}]$ in the training table specifies the number of all training pairs[1]. Thus, the probability $P(rel_{i,j} \wedge range_{i,k})$ is computed as

$$P(rel_{i,j} \wedge range_{i,k}) = \frac{T[range_{i,k}, rel_{i,j}]}{\Sigma_{k,j}T[range_{i,k}, rel_{i,j}]} \tag{5.2}$$

Note that $\forall j,k : P(rel_{i,j} \wedge range_{i,k}) \in [0,1]$ and that $\Sigma_{k,j}P(rel_{i,j} \wedge range_{i,k}) = 1$.

The sum of all cells in a single horizontal dimension k is $\Sigma_j T[range_{i,k}, rel_{i,j}]$. Based on this sum we can compute the probability $P(range_{i,k})$ as

$$P(range_{i,k}) = \frac{\Sigma_j T[range_{i,k}, rel_{i,j}]}{\Sigma_{k,j}T[range_{i,k}, rel_{i,j}]} \tag{5.3}$$

Note that $\forall k : P(range_{i,k}) \in [0,1]$ and that $\Sigma_k P(range_{i,k}) = 1$.

Thus, P is a probability function [78] and can therefore be used as BPA (Definition 4.7) to define USMs (Definition 4.9). Note here that the space complexity of identifying P depends on the number of cells $T[range_{i,k}, rel_{i,j}]$. If we assume that the number of $range_{i,k}$ ranges is constant, then the space complexity depends only on the number of $rel_{i,j}$ sets, which can be maximum $2^{|\Theta_{rel}|} = 2^6$.

[1] Apart from the ones with $sim' =$ `NaN`.

Example 5.1. Computing USMs In Table 5.1 the cell $T[(.9,1],\overset{S}{\approx}]$ specifies that 7 pairs have a similarity degree in the $(.9,1]$ range and their mapping is in set $\overset{S}{\approx}$. The sum of all cells is 339, *i.e.* 339 pairs have been used as training. Thus, the joint probability $P((.9,1] \wedge \overset{S}{\approx})$ is computed as the fraction of the number of pairs in cell $[(.9,1],\overset{S}{\approx}]$ and the number of all pairs, *i.e.* $P((.9,1] \wedge \overset{S}{\approx}) = 7/339$. The sum of all cells in the dimension $(.9,1]$ $\Sigma_j T[(.9,1], rel_j]$ is 10. Thus, $P((.9,1]) = 10/339$ and based on Equation 5.1 we have that $P(\overset{S}{\approx}|(.9,1]) = P((.9,1] \wedge \overset{S}{\approx}) \div P((.9,1]) = 7/399 \div 10/339 = 7/10$. Similarly, $P(\{\overset{S}{\not\sim}\}|(.9,1])) = P(\{\overset{S}{\not\sim}\} \wedge (.9,1]) \div P((.9,1]) = 3/339 \div 10/339 = 3/10$.

Now, consider that in a new matching task there is a pair s_1, s_2 of objects, whose similarity degree falls in the range $(.9,1]$. The USM the expert will produce based on training data in Table 5.1, is $\langle s_1,\ m,\ s_2\rangle$, where

$$m(X) = \begin{cases} P(\overset{S}{\approx}|(.9,1]) & \text{if } X = \overset{S}{\approx} \\ P(\{\overset{S}{\not\sim}\}|(.9,1]) & \text{if } X = \{\overset{S}{\not\sim}\} \end{cases} \tag{5.4}$$

where $P(\overset{S}{\approx}|(.9,1]) = 7/10$ and $P(\{\overset{S}{\not\sim}\}|(.9,1])) = 3/10$.

$\diamond$

In our implementation, the vertical dimension of the training table is split into $1/interval$ intervals, where $1\ mod\ interval = 0, 0 < interval \leq 1$. Each line $k, k \neq 0$ of the training table is associated with a range $(interval \times k, interval \times (k+1)]$, while line 0 of the table is associated with range $[0, interval]$. In Figure 5.1, we have $interval = 0.1$. The value of the *interval* is individual for each expert. The training table in each trainer is specified as a tuple $\langle interval, \langle rel_{i1}, rel_{i2}, \ldots, rel_{in}\rangle\rangle$, where $rel_{ij} \subseteq \{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\emptyset}, \overset{S}{\not\sim}\}, rel_{ix} \cap rel_{jy} = \emptyset, x \neq y$. Table 5.1 illustrates a $\langle 0.1, \langle \overset{S}{\approx}, \{\overset{S}{\not\sim}\}\rangle\rangle$ table.

The training process is performed as follows. Initially, each expert sets the value of each element of the training table to zero. Then, for each training matching task the expert compares each pairs of schema objects p' and produces a similarity degree sim'. The semantic relationship rel' for p' is provided by the user. Based on these values the cell $T[range_{ik}, rel_{ij}]$, with $sim' \in range_{ik}$ and $rel' \in rel_{ij}$ is incremented. Note that only one rel_{ij} of the training table will match the relationship rel' since the horizontal dimension of the table does not include intersecting sets. If the similarity degree produced by the expert is `NaN`, *i.e.* not a number, the expert just discards the pair.

After the training process, the expert, based on the training table, can determine all the necessary probabilities, and produce USMs. Note that in case the expert produces similarity degree `NaN` for a pair of objects s_1, s_2 then the USM resulting from is $\langle s_1, m, s_2 \rangle, m(\Theta) = 1$, which means that the expert expresses its total ignorance about the particular pair.

The above implementation suffers when there are no training pairs that map to specific ranges in the [0,1] range. For example, in training table Table 5.1, there are no training pairs that fall in the (0.8,0.9] range. If the similarity degree of a new pair p' of objects falls in such a range, the expert is not be able to compute the necessary probabilities. In this case the expert shows total ignorance about the mapping of p', *i.e.* produces USM $\langle s_1, m, s_2 \rangle, m(\Theta) = 1$.

However, intuitively even for ranges with no previous training data, it might be possible to say that a set of relationships rel_{i,j_1} is more probable than another set rel_{i,j_2}. This intuition would be based on the training data of the adjacent ranges.

Example 5.2. Ranges with no training data Table 5.1 shows that there are no training pairs in the (0.8,0.9] range. However, we see from the table that as the similarity degree increases the proportion of compatible

to incompatible pair of objects also increases.

First, we can calculate the probabilities for compatibility and incompatibility in all ranges with training data, similar to Example 5.1:

$P(\overset{S}{\not\sim}\|[0,0.1])$	=	0.986	$P(\overset{S}{\sim}\|[0,0.1])$	=	0.014
$P(\overset{S}{\not\sim}\|(0.1,0.2])$	=	0.866	$P(\overset{S}{\sim}\|(0.1,0.2])$	=	0.1333
$P(\overset{S}{\not\sim}\|(0.2,0.3])$	=	0.7826	$P(\overset{S}{\sim}\|(0.2,0.3])$	=	0.2174
$P(\overset{S}{\not\sim}\|(0.3,0.4])$	=	0.666	$P(\overset{S}{\sim}\|(0.3,0.4])$	=	0.333
$P(\overset{S}{\not\sim}\|(0.5,0.6])$	=	0.0	$P(\overset{S}{\sim}\|(0.5,0.6])$	=	1.0
$P(\overset{S}{\not\sim}\|(0.7,0.8])$	=	0.0	$P(\overset{S}{\sim}\|(0.7,0.8])$	=	1.0
$P(\overset{S}{\not\sim}\|(0.9,1.0])$	=	0.3	$P(\overset{S}{\sim}\|(0.9,1])$	=	0.7

For some ranges the probabilities are missing, *e.g.* $P(\overset{S}{\not\sim}|(0.8,0.9])$ and $P(\overset{S}{\sim}|(0.8,0.9])$. In these ranges, the probabilities cannot be defined because there are no training examples.

Looking at the above probabilities, we can see that the probability for incompatibility from the 0.986 value in the [0,0.1] range decreases to 0.3003 in the (0.9,1.0] range, while the probability for compatibility from the 0.014 value in the [0,0.1] range increases to value 1 in the (0.7,0.8] range and value 0.6997 in the (0.9,1] range.

Thus, for a pair of objects that the similarity degree is in the (0.8,0.9] range it is more probable that the pair is compatible rather than incompatible. So the trainer that shows total ignorance for this pair based on the training data is not precise.

◇

To resolve this issue, a trainer can consider the ranges and their conditional probabilities as data points (x, y) and try to fit a curve to these points. In particular,

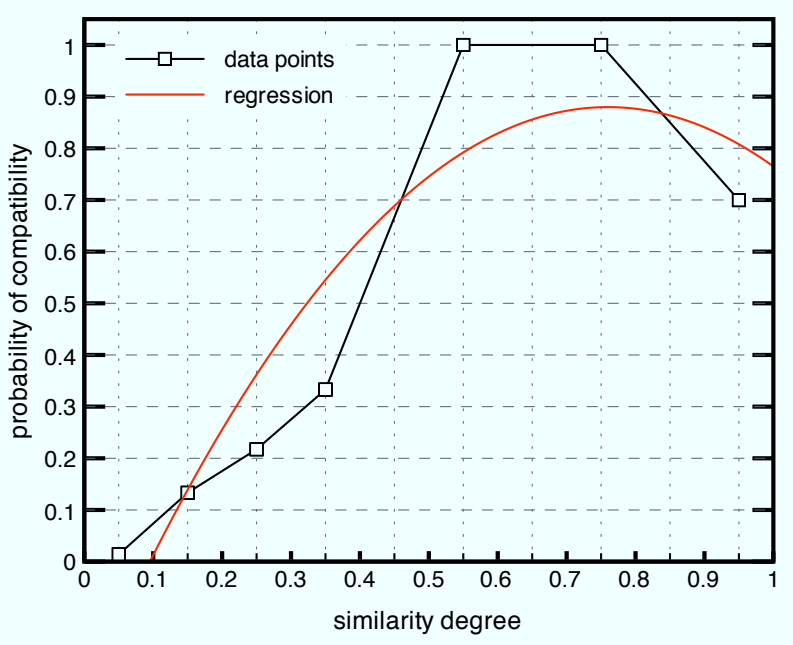


Figure 5.1: Regression on probability $P(\overset{S}{\approx}|range_{ik})$

for each set rel_{ij} of mappings in the horizontal dimension of the training, a distinct curve needs to be computed. For each set of mappings rel_{ij}, a data point $(x, y) = (medium(range_{ik}), P(rel_{ij}|range_{ik}))$ is defined for each combination of $range_{ik}$ and conditional probability $P(rel_{ij}|range_{ik})$. For example, regarding the set $\overset{S}{\approx}$ of the training table Table 5.1 the probability $P(\overset{S}{\approx}|(0.9, 1]) = 0.7$ can be translated into a data point $(0.95, 0.7)$, since $medium((0.9, 1]) = 0.95$. Figure 5.1 illustrates all the data points that can be derived from Table 5.1 for $\overset{S}{\approx}$.

In our implementation, we have used *least-squares regression* analysis to identify the curve. For each set of relationships rel_{ij}, the trainer can identify a curve $f_{rel_{ij}}(x)$ that fits the corresponding data points. The curve $f_{rel_{ij}}$ is identified as the function that minimizes the square error: $\Sigma_{i=1}^{n}(y_i - f_{rel_{ij}}(x_i))^2$. Based on this curve $f_{rel_{i,j}}$, the trainer can compute the conditional probability $P(rel_{ij}|[sim, sim])$ as $P(rel_{i,j}|[sim, sim]) = f_{rel_{ij}}(sim)$ for any $0 \leq sim \leq 1$. In our implementation, we identify a quadratic curve to fit the data points.

Example 5.3. Regression trainer Following on from Example 5.2, the ranges and the relationship probabilities define a series of data points (x, y). Figure 5.1 illustrates the data points for compatibility. The value of x in each point is the mean value of the range this point represents, *e.g.* for range (0.3,0,4] $x = (0.4 + 0.3)/2 = 0.35$. The value of y in each point is the probability of the relationship this point represents, *e.g.* for range (0.3,0.4] $y = P(\overset{\text{S}}{\approx}|(0.3, 0.4]) = 0.333$.

By performing quadratic least-squares regression analysis, we can identify a curve $f(x) = -1.991x^2 + 3.026x - 0.27$, which is also illustrated in Figure 5.1. Thus, while previously we could not calculate the probability of compatibility for the (0.8,0.9] range, using f we can deduce that the probability a pair of object is compatible given that the similarity degree produced for this pair is 0.85, $P(\overset{\text{S}}{\approx}|0.85) = f(0.85) = 0.8636$, which confirms our intuition that any pair with similarity degree in (0.8,0.9] is more probable to be compatible than incompatible.

◇

In our implementation, a *regression* trainer has been implemented using the flanagan[2] package that performs a quadratic least squares regression analysis. Note that we normalize the results of the identified curve, if the probability given by the derived function is out of the [0,1] range:

$$P(rel|[sim, sim]) = \begin{cases} f_{rel_{ij}}(sim) & \text{if } 0 \leq f_{rel_{ij}}(sim) \leq 1 \\ 0 & \text{if } f_{rel_{ij}}(sim) < 0 \\ 1 & \text{if } f_{rel_{ij}}(sim) > 1 \end{cases} \tag{5.5}$$

[2]http://www.ee.ucl.ac.uk/∼mflanaga/java/

5.2 Top-K

The Top-K component of our schema integration tool selects the top-K schema mappings for a matching task. We mentioned in Chapter 4 that the number of possible schema mappings is exponential to the number N of pairs of objects in the matching task. Thus, we do not want to identify all possible schema mappings nor compute their belief or plausibility. Instead, we want to identify only the top-K schema mappings.

In our implementation, the ordering of the schema mappings could be based on either plausibility Pl or belief Bel. We remind the reader that Chapter 4 shows how belief and plausibility of schema mappings can be derived from USMs. The rest of this section talks about deriving top-K mappings based on belief, but the same process can be applied for top-K mappings based on plausibility. Our implementation is based on Lemma 1 (Section 4.7), which shows that the belief of a schema mapping can be identified without computing the BPA M_{Match} for the schema mappings.

5.2.1 Processing the uncertain schema mapping

The input to the Top-K component is an uncertain schema mapping u-Match, *i.e.* a list of N USMs (Def. 4.10) one for each of the N pairs of objects compared. The first step of the Top-K component is to process each one of these N USMs u-Match[i] and derive the belief Bel for each semantic relationship for each pair p_i of objects. These beliefs are crucial according to Lemma 1, which states that the belief of a schema mapping is equivalent to the product of the beliefs of the semantic relationships it is composed of.

In the second step, the semantic relationships for each pair p_i are ranked based on their belief. This process produces a list l_i of semantic relationships for each pair p_i of objects. The list stores the relationships from the highest ranking relationship

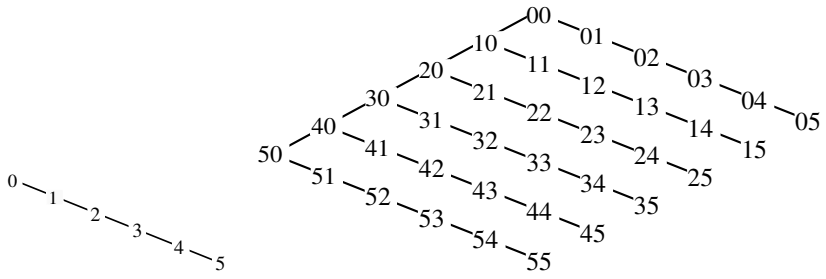
$l_i[0]$ to the lowest ranking relationship $l_i[NR-1]$, where NR is the number of possible relationships. Hence, the j-th ranked semantic relationship for pair p_i is $l_i[j]$. Our implementation allows for $NR = 6$, since we support 6 distinct semantic relationships (Definition 4.3).

Now, we can perform K iterations to identify the top-K schema mappings. In each iteration, we select the schema mapping with the next highest belief. For example, in the first iteration the schema mapping with the highest belief is the one that is composed of the semantic relationships at rank 0 for all pairs, $l_i[0]$. In the second iteration, the semantic relationship of one of the pairs, *e.g.* p_j, needs to be altered to $l_j[1]$. The pair p_j that is selected is the one that causes the smallest change in the belief of the schema mapping, *i.e.* $\forall i, i \neq j : |Bel(l_j[0]) - Bel(l_j[1])| \leq |Bel(l_i[0]) - Bel(l_i[1])|$. The next section shows this process in detail.

5.2.2 Exhaustive Top-K

To efficiently compute the top-K schema mappings for a particular matching task, we use a tree structure. Each node of the tree represents one schema mapping and each possible schema mapping corresponds to a node in the tree. Thus, we call this tree structure **exhaustive top-K tree**.

Each schema mapping sm in this exhaustive top-K tree, is abbreviated using a N-digit string. Each position i of the string corresponds to pair p_i. The value $sm[i]$ of the string at position i is the rank of the semantic relationship selected for pair p_i in sm, which corresponds to the semantic relationship $l_i[sm[i]]$. For example, assume that $N = 2$ and schema mapping sm' has string abbreviation 02. The relationship that has been selected for pair p_1 in sm' is the relationship with rank $sm'[1] = 0$ and for pair p_2 is the relationship with rank $sm'[2] = 2$. The belief of each sm is computed as the product of the beliefs of the semantic relationships sm is composed of (Lemma 1). Thus, $Bel(sm) = Bel(l_1[sm[0]]) \times \ldots \times Bel(l_N[sm[N]])$. For example,



(a) $N = 1$ (b) $N = 2$

Figure 5.2: Exhaustive top-K trees for $NR = 6$

regarding the schema mapping sm' mentioned above, we have $Bel(02) = Bel(l_1[0]) \times Bel(l_2[2]])$.

Each node of the exhaustive top-K tree has an associated belief, which is the belief of the schema mapping the node corresponds to. The tree is structured to ensure that each node has belief equal or greater than the belief of its descendants.

For example, Figure 5.2(a) illustrates the complete exhaustive top-K tree for $N = 1$ and $NR = 6$. The root of this tree represents schema mapping, sm_0, with string abbreviation 0. To produce the descendants of the root, the 1-th position of the schema mapping is incremented. The only child of sm_0 represents schema mapping sm_1 with string abbreviation 1. The belief of sm_1 is less than or equal to the belief of sm_0, since by definition of l_1 we have that $Bel(l_1[sm_0[1]]) = Bel(l_1[0]) \geq Bel(l_1[sm_1[1]]) = Bel(l_1[1])$ and $Bel(sm_1) = Bel(l_1[1])$, $Bel(sm_0) = Bel(l_1[0])$, thus $Bel(sm_1) \leq Bel(sm_0)$.

The exhaustive top-K tree is gradually built as follows. In the first iteration, the root of the tree is identified as the schema mapping whose string abbreviation contains only zeroes. This mapping, where each pair has been associated with semantic relationship with rank 0, has the highest belief and it is the first mapping out of the top-K. For example, in Figure 5.2(b) the schema mapping 00 is the root of the top-K tree for $N = 2$ and $NR = 6$.

Before moving on to the next iteration, the children of the root node must be identified. The children of the root node are stored in a list of *current leaf nodes* together with their beliefs. One of these leaf nodes will be the schema mapping with the next highest belief, *i.e.* the second schema mapping in the top-K.

In general, the children of each node are defined by incrementing a single position in the string abbreviation of the mapping of the parent node. For example, the children of node 00 are: 10 and 01. If there is a child node which has been produced by incrementing the i position of its parent, then the descendants of this child node cannot update their first $i-1$ positions. This constraint ensures that there are no two nodes in the tree that correspond to the same schema mapping. For example, in Figure 5.2(b) the descendants of node 01, which has been produced from node 00 by incrementing position 2, cannot update the first $2-1=1$ positions. Therefore, the 01 has only one child, node 02. Similarly, for 02 and the rest of the 01 descendants, *i.e.* 03, 04 and 05, the first 1 positions cannot be incremented.

Now, at each iteration i, $i \leq K$, we search through the list of current leaf nodes and identify the schema mapping with the highest belief. This will be the mapping at rank i. Before we start iteration $i+1$, we identify the children of the i-th mapping and add them in the list of current leaf nodes.

For example, in Figure 5.2(b) after the identification of the root node the list of current leaf nodes contains nodes 01 and 10. Now assume that in the second iteration node 10 has the highest belief and it is the one selected. The list of current leaf nodes is now updated to contain also the children of 10, *i.e.* it contains in total nodes $20, 11$ and 01. One of these nodes is going to be third mapping of the top-K.

5.2.3 Local Flatlines

Even though the exhaustive top-K approach identifies all possible schema mappings, it produces **local flatlines** that may delay the identification of a preferable mapping.

To illustrate this issue, assume that in a matching task we are just interested in compatibility and incompatibility mappings. Assume that the task contains N pairs of objects and that v pairs, $p_1, \ldots, p_v, v \leq N$, have identical USMs and thus identical beliefs for compatibility and incompatibility *i.e.* $Bel_{p_i}(\overset{S}{\approx}) = Bel_{p_j}(\overset{S}{\approx})$ and $Bel_{p_i}(\overset{S}{\not\sim}) = Bel_{p_j}(\overset{S}{\not\sim})$ for all $1 \leq i \leq v, 1 \leq j \leq v, i \neq j$. A reason this may happen is because the similarity degrees for these v pairs fall in the same range in the training table and therefore the v pairs have identical USMs. Additionally, assume that amongst the N pairs, the v aforementioned pairs are the ones that the tool is least certain about their semantic relationships, *i.e.* $|Bel_{p_i}(\overset{S}{\approx}) - Bel_{p_i}(\overset{S}{\not\sim})| < |Bel_{p_j}(\overset{S}{\approx}) - Bel_{p_j}(\overset{S}{\not\sim})|, 1 \leq i \leq v, v < j \leq N$. Note here that in an interactive setting, where the user validates the produced semantic mappings, the identification of these v pairs is crucial since these pairs can be the first that are examined and corrected if necessary. This is an advantage of our approach and a result of the explicit representation of the uncertainty of the tool on each semantic mapping.

Now, the v pairs of objects specify a sub-graph in the exhaustive top-K tree, for each possible configuration of the $N - v$ pairs. Each such sub-graph contains 2^v schema mappings, one for each possible alternative configuration of the v pairs of objects, while the configuration of any other pairs is constant. In each sub-graph, the beliefs of the mappings that belong to the same depth of the sub-graph are identical. Therefore, the schema mappings at each depth of each sub-graph have consecutive ranks in the exhaustive top-K approach. When all these mappings are visited a local belief flatline is produced. It is a flatline because the belief of the schema mappings does not change even though the schema mappings change.

Example 5.4. Exhaustive Top-K delayed by local flatlines

In a matching task, there are four pairs of schema objects: $p_1 \equiv \langle s_{11}, s_{21} \rangle$, $p_2 \equiv \langle s_{11}, s_{22} \rangle$, $p_3 \equiv \langle s_{12}, s_{21} \rangle$ and $p_4 \equiv \langle s_{12}, s_{22} \rangle$. The single expert used, calculates the similarity degrees for the four pairs, $sim(p_1) = 0.95$,

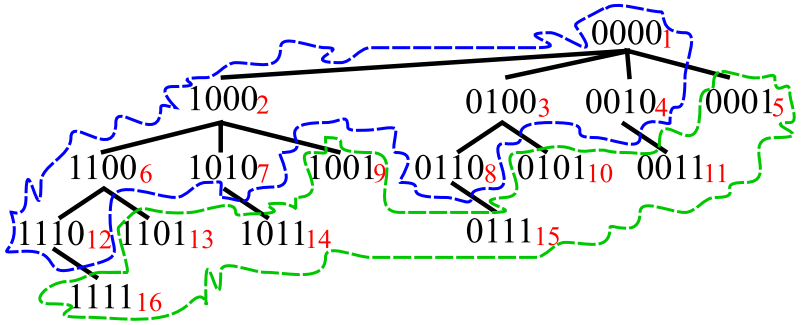


Figure 5.3: Exhaustive Top-K with 4 pairs and $NR = 2$

$sim(p_2) = 0.95$, $sim(p_3) = 1$, $sim(p_4) = 0.3$, and based on the training data, which are illustrated in Table 5.1, outputs the following USMs:

$$\langle s_{11}, m_{5.1}, s_{21} \rangle,$$

$$\langle s_{11}, m_{5.1}, s_{22} \rangle,$$

$$\langle s_{12}, m_{5.1}, s_{21} \rangle,$$

$$m_{5.1}(\{\overset{\text{S}}{=}, \overset{\text{S}}{\subset}, \overset{\text{S}}{\supset}, \overset{\text{S}}{\cap}, \overset{\text{S}}{\not\cap}\}) = 0.7$$

$$m_{5.1}(\{\overset{\text{S}}{\not\sim}\}) = 0.3$$

$$\langle s_{12}, m_{5.4}, s_{22} \rangle$$

$$m_{6.7}(\{\overset{\text{S}}{=}, \overset{\text{S}}{\subset}, \overset{\text{S}}{\supset}, \overset{\text{S}}{\cap}, \overset{\text{S}}{\not\cap}\}) = 0.2174$$

$$m_{6.7}(\{\overset{\text{S}}{\not\sim}\}) = 0.7826$$

In this matching task, we are interested only in compatibility and incompatibility relationships. Therefore, there are 2^4 possible schema mappings. The exhaustive top-K tree for all these schema mappings is illustrated in Figure 5.3. In each schema mapping in the figure, the first position corresponds to pair p_1, the second position corresponds to p_2, the third position to p_3 and the fourth position to p_4. Each schema mapping in the figure is annotated with its rank, *e.g.* the root 0000 is annotated with rank 1. After

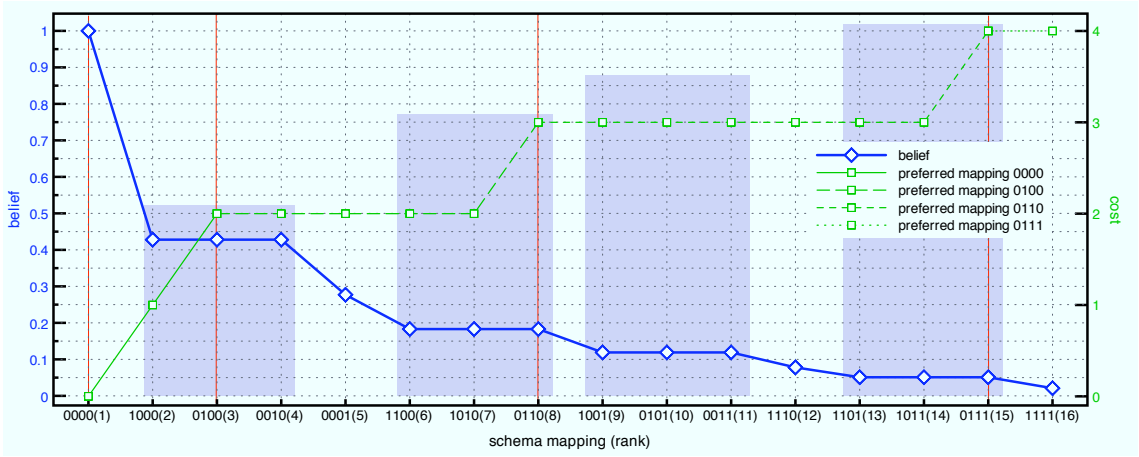


Figure 5.4: Exhaustive Top-K defining local flatlines

the root in rank 1, the schema mappings in the next 3 ranks are derived by altering the configuration of one of the first pairs of objects, since these pairs are the ones that the tool is least certain about: $|Bel_{p_1}(\overset{S}{\sim}) - Bel_{p_1}(\{\overset{S}{\gamma}\})| = |Bel_{p_2}(\overset{S}{\sim}) - Bel_{p_2}(\{\overset{S}{\gamma}\})| = |Bel_{p_3}(\overset{S}{\sim}) - Bel_{p_3}(\{\overset{S}{\gamma}\})| = |0.7 - 0.3| = 0.4 < |Bel_{p_4}(\overset{S}{\sim}) - Bel_{p_4}(\{\overset{S}{\gamma}\})| = |0.2174 - 0.7826| = 0.5652$. Figure 5.4 illustrates the belief of each schema mapping of the 2^4 schema mappings. The schema mappings in the figure are in ranking order.

In Figure 5.3, a sub-graph of 2^3 schema mappings is defined for each possible configuration of pair p_4. The first sub-graph contains mappings 0000, 1000, 0100, 0010, 1100, 1010, 0110 and 1110, where the configuration of p_4 is 0. The second sub-graph contains mappings 0001, 1001, 0101, 0011, 1101, 1011, 0111 and 0111, where the configuration of p_4 is 1. The mappings that belong to the same depth of a sub-graph have identical beliefs. For example, the mappings 1000, 0100 and 0010 in ranks 2, 3, and 4 respectively, have belief 0.428, *e.g.* $Bel(1000) = \frac{Bel_{p_1}(\{\overset{S}{\gamma}\}) \times Bel_{p_2}(\overset{S}{\sim}) \times Bel_{p_3}(\overset{S}{\sim}) \times Bel_{p_4}(\{\overset{S}{\gamma}\})}{Bel_{p_1}(\overset{S}{\sim}) \times Bel_{p_2}(\overset{S}{\sim}) \times Bel_{p_3}(\overset{S}{\sim}) \times Bel_{p_4}(\{\overset{S}{\gamma}\})} = 0.428$. Note that the belief of the mapping is normalized so that the belief of mapping 0000 is 1. The three schema mappings 1000, 0100 and 0010 define a local flatline, which is illustrated in Figure 5.4 as a shaded area.

The figure also illustrates the selection cost of a preferable mapping in an

interactive setting in terms of the number of pairs of objects compared, as defined in Section 4.7.1. First of all, assume that the user's preferred mapping is 0111 for this task. In step (a), the user compares the first two schema mappings in top-16, *i.e.* the mappings 0000 and 1000 at rank 1 and rank 2 respectively. The cost of this comparison is the cost of comparing a single pair of objects, in particular pair p_1, since all other pairs have identical relationships. Now, mapping 0000 is more correct than 1000 according to this user, therefore the mapping 0000 is selected by the user and is carried on to the next schema mapping comparisons. Effectively, by selecting mapping 0000 over 1000 it shows that the user prefers for p_1 the relationship 0 rather than the relationship 1.

In step (b), mapping 0000 is compared with the mapping 0100 at rank 3. This cost of this comparison is the cost of comparing the two relationships of p_2. Thus, an additional pair to p_1 is now compared increasing the number of pairs compared and increasing the selection cost to two. Mapping 0100 is better than 0000, therefore this mapping is carried on to the next steps. Effectively, the selection of 0100 over 0000 shows that the user prefers the relationship 1 for p_2 rather than the relationship 0.

Now, based on the user's previous selections, the tool knows that the mappings that the user is interested in are mappings of the form 01__, *i.e.* they start with[3] 01. All other mappings can be skipped and not displayed to the user, who would not need to compare them. Therefore, the overall user effort and the overall selection cost is reduced.

Thus, in step (c), the next four mappings 0010, 0001, 1100 and 1010 are skipped and do not affect the cost of selection. Mapping 0110, however, is of the form 01__. The user compares mapping 0100 with 0110 and decides to select mapping 0110, which is more correct. The cost of this comparison

[3]Note that in an example with $NR = 6$, the mappings the user would be interested in would be of the form 01__,21__, *etc.*

is another pair of relationships to be compared for pair p_3. Thus, the cost increases to three pairs.

Similarly, now all the mappings of interest are of the form 011_, which allows skipping the next 6 mappings. In step (d), when mapping 0111 is reached, it is compared against 0110 and it is selected. The user has identified the correct mapping for this task. The cost is increased to 4.

Finally, in step (e), mapping 1111 is skipped and not displayed to the user since it does not follow the current selected form 0111.

◇

As we see from the above example, local flatlines do not affect the cost of the search process in an interactive setting, *i.e.* when the user gradually specifies which schema mappings out of the top-K she prefers until she reaches a preferable mapping. The tool based on the user's preferences can guide the search process and avoid schema mappings which are irrelevant to the user.

However, in a fully automatic setting where there is no user intervention, irrelevant schema mappings cannot be avoided. In the example above, in a fully automatic setting where the top-16 mappings are required, the tool has to consider all 16 mappings shown in Figure 5.4. Thus, the tools has to pass four local flatlines. The same mapping 1111 is reached in 16 steps (as many as the mappings), while in the interactive setting the 16 mappings are examined in five steps (mappings 0000, 1000, 0100, 0110 and 0111) based on user preference. Thus, local flatlines in a fully automatic setting could delay the identification of the preferable mapping. This is even more problematic since the number of schema mappings in a local flatline increases exponentially as shown next.

Assume again that a matching task contains N pairs of objects of which v pairs have identical beliefs for compatibility and incompatibility. As aforementioned, these v pairs specify a subgraph in the exhaustive top-K tree and at each depth of this

subgraph a local flatline is defined. We call the number of schema mappings at each depth of the subgraph, the *size of the local flatline*.

Each schema mapping at depth i of the sub-graph, when compared to the root schema mapping has i changes on the configuration of the v pairs. The root configuration contains only zeroes, therefore each schema mapping at depth i has i ones and $v-i$ zeroes. To calculate the number of possible configurations with i ones and $v-i$ zeroes we need to calculate the combinations $\binom{v}{i}$. Thus, the size of the local flatline at depth i for the v pairs is

$$size(i) = \binom{v}{i} = \frac{v!}{(v-i)!\ i!} \tag{5.6}$$

For example, consider that $v = 3$ and depth $i = 2$. At depth $i = 2$ there are going to be 2 out of the 3 pairs with configuration one and one pair with configuration zero. The number of such mappings and thus the size of the local flatline is $\binom{3}{2} = 3$. In Example 5.4, where $v = 3$ (p_1, p_2, p_3), in depth $i = 2$ of the sub-graph with root 0000, the $\binom{3}{2}$ possible mappings are 1100, 1010 and 0110, *i.e.* two out of the first three pairs have configuration one, and these mappings constitute a local flatline as shown in Figure 5.4. If we increment v, $v = 4$, then for $i = 2$ there are going to be 2 out of the 4 pairs with configuration one and 2 pairs with configuration zero, *i.e.* the local flatline size if $\binom{4}{2} = 6$. For $v = 5$ and $i = 2$ the size is $\binom{5}{2} = 10$, and for $v = 6$ it is $\binom{6}{2} = 15$.

The graph in Figure 5.5, which is derived by Equation 5.6, illustrates that the size of a local flatline at each depth increases exponentially as the number of pairs v increases.

Thus, in a fully automatic setting, a top-K method which avoids local flatlines would be beneficial.

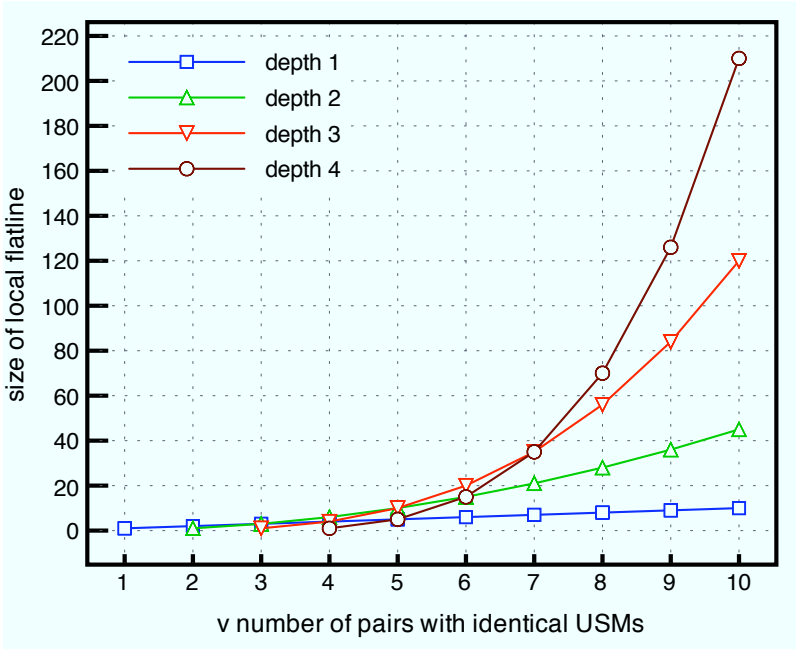


Figure 5.5: Exponentiality of the size of local flatlines

5.2.4 Truncated Top-K

In the **truncated top-K** approach, the pairs of objects that have identical USMs are grouped together. This approach is used in a fully automatic setting, therefore the user cannot intervene nor differentiate between the pairs in a group. In a matching task with N pairs of objects, there are N_g groups of pairs, $G_1, \ldots, G_{N_g}, N_g \leq N$. The number of pairs in each group G_i is $|G_i|$. Each two pairs of objects, $p_i, p_j, 1 \leq i \leq N, 1 \leq j \leq N, i \neq j$ that belong to the same group of pairs, G_l, *i.e.* $p_i \in G_l, p_j \in G_l$, have identical USMs.

As in the exhaustive top-K approach, each schema mapping sm in the truncated top-K approach is represented as a string of length N_g. Each position i in sm now represents the semantic relationship of group $G_i, i \leq N_g$. Thus, all pairs of objects in the same group have the same semantic relationship in sm. The position i in sm is given a value $0,\ldots,NR-1$, which specifies now the rank of the relationship selected for group G_i in sm. Since all pairs in G_i have identical USMs, the ranking of the semantic relationships amongst these pairs is identical. Group G_i uses this

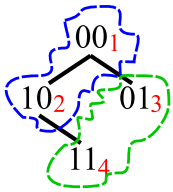


Figure 5.6: Truncated Top-K with 2 groups of pairs and $NR = 2$

ranking. In the truncated top-K approach, the process to derive the top-K schema mappings is the same as before, *i.e.* the top-K tree is gradually built but for each configuration of the groups $G_1, \ldots, G_{N_g}$.

In the truncated approach, the sub-graphs that were previously causing the local flatlines are also truncated, *e.g.* in the case where $NR = 2$ the sub-graph with $2^{|G_i|}$ different nodes is replaced with a truncated version with just two nodes, a root and a leaf; the root of the truncated sub-graph represents exactly the same mapping as the root node in the exhaustive sub-graph, and the leaf node of the truncated sub-graph represents exactly the same mapping as the leaf node in the exhaustive sub-graph.

The drawback of the truncated top-K is that not all possible schema mappings for a particular task are identified. The advantage of this approach is that it is not delayed by local flatlines.

Example 5.5. Truncated Top-K

Following on from Example 5.4, in the truncated top-K approach pairs p_1,p_2 and p_3 form group G_1, since they have identical USMs, and pair p_4 forms group G_2. The truncated top-K tree contains 2^2 mappings and is illustrated in Figure 5.6.

Each sub-graph in the exhaustive approach, Example 5.4, is replaced by a truncated version of it. Mappings 00 and 10 in Figure 5.6 constitute the truncated version of the sub-graph in Figure 5.3 with root node 0000 and

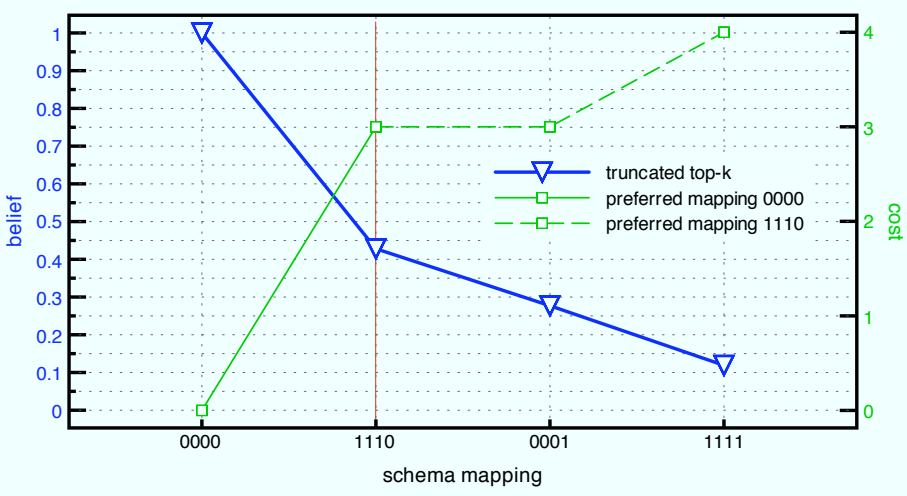


Figure 5.7: Truncated Top-K beliefs

leaf node 1110. Mapping 00 corresponds to mapping 0000 in the exhaustive top-K approach and mapping 10 corresponds to 1110. Mappings 01 and 11 constitute the truncated version of the sub-graph in Figure 5.3 with root node 0001 and leaf node 1111. Mapping 01 corresponds to 0001 and mapping 11 corresponds to 1111.

There are 12 more mappings in the exhaustive top-K tree, which are not considered in the truncated version.

Figure 5.7 illustrates the beliefs of the 2^2 mappings identified in the truncated approach. The schema mappings in the figure are in ranking order and have been expanded into their exhaustive top-K counterparts.

We can see that Figure 5.7 is free of local flatlines. Additionally, we see that a reasonable mapping 1110, which is 50% correct (0111 is the user's preferred mapping) is just the second mapping in the truncated top-4, while it is the 12-th mapping in the exhaustive top-16 mappings.

◇

5.3 Experimental Evaluation

In this section, an experimental evaluation of our matching implementation is presented. First, we want to evaluate how the introduction of USMs in the matching process affects the quality of the derived schema mapping. In order to examine this issue we compare the schema mapping derived by the matching algorithms when they produce similarity degrees (see Section 5.1.2) compared to the top-1 schema mapping derived using the same algorithms extended to produce USMs. This evaluation step will show whether the introduction of uncertainty in matching in the form of USMs affects top-1 matching. Our experiments show that top-1 matching is in fact improved with the use of USMs.

Additionally, we want to evaluate the improvement we gain in the quality of the schema mapping when looking at the top-K schema mappings rather than just the top-1 mapping. This evaluation step will show whether there is an advantage in examining top-K mappings rather that just the top-1 mapping. Our experiments confirm our intuition that better schema mappings than the top-1 schema mapping are available in the top-K mappings.

In the rest of this section, we first describe the data set that we use to perform our evaluation. We have adopted the data set also used in [73] for the evaluation of a matching implementation because it supplies us with data that can be used for training our matching experts to produce USMs. Then we list the matching experts we experimented with. Our matching experts have been configured to assign probability mass to two sets of semantic relationships $\{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\emptyset}\}$ and $\{\overset{S}{\not\sim}\}$ since the data set we have adopted does not specify more specific semantic relationships to pairs of objects, *e.g.* $\{\overset{S}{=}\}$. or $\{\overset{S}{\cap}\}$, *etc.* Then we discuss the evaluation metric, *accuracy*, used to measure the quality of schema mappings.

Having defined the evaluation metric, we have compared our two proposed approaches (exhaustive and truncated top-K) for the derivation of the top-K schema

mappings (Section 5.3.4), and we have explained when each approach can guarantee improvement on the accuracy of the schema mappings produced. In Section 5.3.5, we compare our matching experts based on whether they use regression to define USMs and we show that using regression produces schema mappings of higher accuracy on average for our experts. Finally, we describe our experiments and our results.

5.3.1 Data Set

The evaluation of our schema matching implementation requires a set of schema matching tasks T where manual matching has been performed by a user u in each task $t \in T$. Each task $t \in T$ specifies an integration of two schemas, S_{t1} and S_{t2}, with sets of schema objects SO_{t1} and SO_{t2} respectively. The user u must have defined the schema mapping m_{tu} that holds between S_{t1} and S_{t2} according to her perspective.

The schema matching tasks that we used in the evaluation are the tasks introduced in [73]. Some schemas in these tasks have been borrowed from research papers and others have been derived from data used on websites such as Amazon.com and Yahoo.com. In our experiments we used 8 tasks. Both XML and relational schemas are included in these tasks.

Manual matching in each task $t \in T$ has been performed by seven different users $u_1, \ldots, u_7$, from the Stanford Database Group. Thus, for each task t we have seven different versions of Match, *e.g.* for task t_1, user u_1 has specified the schema mapping m_{11} between S_{11} and S_{12}, user u_2 has specified schema mapping m_{21}, *etc.* Interestingly, as noted in [73], no two users could agree on the Match result for any given task, *i.e.* for any users u_x and u_y and any task t, the schema mappings m_{xt} and m_{yt} are different. This enforces our idea of uncertain semantic mappings and the need for training the Match experts for each user. One last point regarding the data

training set size	1	2	3	4	5	6	7
#num experiments	$\binom{7}{1} = 7$	$\binom{7}{2} = 21$	$\binom{7}{3} = 35$	$\binom{7}{4} = 35$	$\binom{7}{5} = 21$	$\binom{7}{6} = 7$	$\binom{7}{7} = 1$

Table 5.2: Number of experiments run for each target task and each user, depending on the size of the training set

set is that the users have specified only compatibility (match) and incompatibility (no-match) semantic mappings.

In our implementation, the schema matching tasks had to be partitioned into tasks used for training and tasks used for the experimental evaluation of our tool, which we call target tasks. We cannot use all the tasks for training since this would produce bias towards the training set, *i.e.* our tool would perform well for the trained tasks but would not be able to deal with new matching tasks as well. This is the well-known, in Machine Learning, problem of **overfitting** [77].

The overfitting problem can be resolved using **cross-validation** [77]. In particular, we applied a version of k**-fold cross-validation**. In k-fold cross-validation, the T tasks are split into k disjoint sets, $T_1, \ldots, T_k$. Each set T_i contains $\frac{|T|}{k}$ tasks, where $|T|$ specifies the size of set T, *i.e.* $|T_i| = \frac{|T|}{k}, 1 \leq i \leq k$. Then the experiments are run k times (k folds) each time using a different set of tasks T_i as target tasks. The remaining set of tasks, $T_j, 1 \leq j \leq k, j \neq i$, are put together to produce the training tasks. Thus, the size of the training set in each experiment is $|T| - |T_i| = |T| - \frac{|T|}{k}$. The results of the k experiments are then averaged.

In our version of k-fold cross-validation, we do not want a constant training set size. Instead, we want to see how our implementation performs as the training set size increases from size 1 up to size $|T| - 1$. Additionally, we want to examine all possible combinations of training tasks and not be restricted to the k sets randomly selected initially in k-fold cross-validation. For example, in a 2-fold cross-validation with $|T| = 8$, where the two disjoint set of tasks are $T_1 = \{t_1, t_2, t_3, t_4\}$ and $T_2 = \{t_5, t_6, t_7, t_8\}$, target task t_1 will be evaluated using only the training tasks of T_2;

tasks t_2, t_3, t_4 are not going to be included in the training. Finally, in our version of k-fold cross-validation, we increase the training set size from size 1 up to size $|T| - 1$, which is similar to performing k-fold cross-validation for increasing values of k. In the last case where the training set size is $|T| - 1$, the value of k is $|T|$. In our particular data set taken from [73], we have $|T| = 8$, and the maximum size of the training set is $|T| - 1 = 7$.

We have conducted the experiments as follows. Each experiment is using a single task t_j as a target task. Each user's u_i schema mapping for this task m_{ij} is used for the evaluation of our implementation. The rest of user's u_i schema mappings for the remaining tasks, $m_{ik}, 1 \leq k \leq 8, k \neq j$, are used as training data. At each stage of the experiments, we increase the size $|TS|$ of the training set TS. There are 7 stages, since this is the maximum size of the training set we can have. In the first run, the size of the training set is $|TS_1| = 1$, meaning that only one of the $m_{ik}, 1 \leq k \leq 8, k \neq j$ is used as training data. In order to avoid overfitting, we run the same experiment, *i.e.* with the same target mappings m_{ij}, but using different training data sets and then we average the results. For example, if m_{i1} is the target schema mapping, then the experiment in stage 1 is first run using training data $TS_1 = \{m_{i2}\}$, then using $TS_1 = \{m_{i3}\}$, *etc.* This way we have $\binom{7}{1} = 7$ combinations of training data, which produce 7 different experiments for stage 1. The results of these experiments are then averaged. In the next stage, stage 2, the size of the training set is incremented, $|TS_2| = 2$. In this stage we have $\binom{7}{2}$ combinations of training data, *i.e.* 21 experimental results, which we average. The same process is executed up to training size $|TS_7| = 7$, which specifies only one possible experiment. Table 5.2 shows in detail the number of experiments ran for each target task for one user as the training set size increases.

5.3.2 Configuration

Due to the fact that only two matching tasks in our data set contain sample instances of the schema objects and therefore the training process would be limited, we decided not to include match experts that perform instance-based comparison of schema objects in our experiments.

The experts we experimented with are the WordNet expert, the Similarity Flooding expert and the Rondo String Matching expert. In addition, we used hardcoded combinations of these experts, where the algorithms of these experts are combined into a single algorithm. These *hybrid* experts are: Similarity Flooding/Rondo, where the initial similarity degrees for the Similarity Flooding algorithm are provided by the Rondo String Matching algorithm, and Rondo/WordNet, where names are tokenized based on the Rondo String Matching algorithm and then tokens are compared using the algorithm of the WordNet expert. Finally, we used combination of experts according to our proposed framework, where the USMs of the experts are combined using our aggregation method (Section 4.4). These experts are: Similarity Flooding + Rondo, Similarity Flooding + WordNet and Similarity Flooding + Rondo/WordNet. These three experts perform both structural and name matching.

Since the users of our data set have only specified compatibility and incompatibility mappings, our experts have been configured to identify just two sets of mappings: $\overset{S}{\sim}$ and $\{\overset{S}{\not\sim}\}$. We used both types of trainers, non-regression and regression based, in our experiments in both cases using $interval = 0.1$. Thus, the training tables used in our experiments are of type $\langle 0.1, \langle \overset{S}{\sim}, \{\overset{S}{\not\sim}\}\rangle\rangle$. Finally, we used belief to order the top-K schema mappings.

5.3.3 Evaluation Metric

In order to evaluate the performance of our Match component and the quality of the schema mappings it produces, we can measure the effort the user is required to spend to transform a schema mapping Match into the user-intended schema mapping [73]. User-effort is measured by counting the number of semantic mappings that need to be altered on the automatically generated schema mapping Match. Note that this user-effort is not the same as the cost of correcting a schema mapping $Cost_{fix}$ presented in Section 4.7.1. $Cost_{fix}$ also considers the user-effort required to identify the correct semantic relationship for each pair before transforming Match to the user-intended schema mapping.

Let: (1) n be the number of compatibility mappings proposed by the tool, (2) $c, c \leq n$ the number of correct compatibility mappings identified by the tool, and (3) m the number of compatibility mappings in the user-intended result. The value $f = n - c$ is the number of false positives mappings, *i.e.* mappings that falsely have been identified as compatible. The value $m - c$ is the number of false negative mappings, *i.e.* mappings that falsely have been identified as incompatible.

If the user performs the whole matching process manually, then the default initial mapping can be considered to be the one where all pairs are incompatible. In this case m semantic mappings need to be altered to translate this mapping to the user-intended one. The user effort in this case is m/m = 1.

If the user first employs the automatic matching process, which produces schema mapping Match, and then corrects the false positives and false negatives of Match, she would require to alter $f + (m - c)$ semantic mappings. Thus, the amount of work needed for transforming an automatically generated schema mapping compared to transforming the default schema mapping is given by the ratio:

$$\frac{f + (m - c)}{m} = \frac{(n - c) + (m - c)}{m}$$

Now, we can measure the quality of a automatically generated schema mapping by calculating the amount of work it saves the user from doing, *i.e.* :

$$A = \frac{m}{m} - \frac{f + (m - c)}{m} = 1 - \frac{(n - c) + (m - c)}{m} \tag{5.7}$$

This metric is called **accuracy** and it is introduced in [73]. In a perfect schema mapping where $n = m = c$, the user-effort required after the automatic matching is 0 and accuracy is 1.

If more than half of the semantic mappings of an automatically generated schema mapping are incorrect, *i.e.* if $c < n/2$, then accuracy is negative, meaning that the user has to do more work than if she was to do the whole process from scratch.

A criticism on the accuracy metric is that it does not take into consideration the number of pairs in a schema matching task. For example, assume a task where there are 5 pairs of objects two of which are compatible, *i.e.* $m = 2$. A schema mapping detects one of these pairs without any false positive pairs, *i.e.* $n = c = 1$. In this case, the schema mapping identifies correctly one of the two compatible pairs between the five pairs in total. Accuracy is 0.5. Now, assume that there are 50 pairs of objects, and still $m = 2$ and $n = c = 1$. In this task, the schema mapping managed to identify one of the two compatible pairs between the 50 in total pairs, which is probably more difficult than in the previous task with 5 pairs. However, accuracy is still 0.5.

Nonetheless, in our evaluation we are still going to use the accuracy metric since it has been previously used in well-known research papers [73, 33]. In addition, all other existing metrics [32] face the same aforementioned problem.

5.3.4 Comparison of Exhaustive and Truncated Top-K based on accuracy

Based on the accuracy metric, we can compare the exhaustive and truncated top-K methods.

At any matching task, there are N pairs of objects, m of which are compatible according to a user. Based on the USMs for these N pairs, an exhaustive top-K tree is defined.

Assume again, as in Section 5.2.3, that we are just interested in compatibility and incompatibility relationships and that v pairs of objects that have identical beliefs for compatibility and incompatibility are the pairs that the tool is least certain about. As it has been mentioned, these v pairs specify a sub-graph in the exhaustive top-K tree for each possible configuration of the $N - v$ pairs, and that at each depth of this sub-graph a local flatline is defined. Assume that the root of such a sub-graph does not represent the user-intended schema mapping, *i.e.* the accuracy A_0 at the root is not 1. Then, the maximum accuracy A_i at each depth i of the sub-graph will be improving, $A_i = A_0 + i/m$, until the user-intended configuration for the v pairs of objects is reached.

Proof. Assume that at the root of each subgraph defined by the v pairs c_v out of v pairs concerned are true positives and f_v out of the v pairs are false positives. Additionally, assume that in the remaining $N - v$ pairs, c_{N-v} are true positives and f_{N-v} are false positives. Thus, at the root of the tree there are in total $f = f_v + f_{N-v}$ false positives and $c = c_v + c_{N-v}$ true positives, which result in accuracy (Equation 5.7) :

$$A_0 = 1 - \frac{(f_v + f_{N-v}) + (m - (c_v + c_{N-v}))}{m} \tag{5.8}$$

At each depth i of the subgraph, the semantic mapping on i of the v pairs is altered compared to their semantic mappings at the root of the tree. The best accuracy will be achieved if all of these i changes are correct changes, *i.e.* all i changes increase the true positives c'_v and/or decrease the false positives f'_v: $c'_v = c_v + x$ and $f'_v = f_v - y$, where $x + y = i$. Thus, the best accuracy at depth i is:

$$\begin{aligned} A_i &= 1 - \frac{(f'_v + f_{N-v}) + (m - (c'_v + c_{N-v}))}{m} = \\ &= 1 - \frac{(f_v - y + f_{N-v}) + (m - (c_v + x + c_{N-v}))}{m} \\ &= 1 - \frac{(f_v + f_{N-v}) + (m - (c_v + c_{N-v})) - (y + x)}{m} \\ &= A_0 + \frac{i}{m} \end{aligned}$$

□

Thus, exhaustively traversing each subgraph identified by the v pairs of objects guarantees an improvement $\frac{i}{m}$ in accuracy at each depth of the subgraph, until the maximal accuracy for the specific configuration of the $N - v$ pairs is reached.

In the truncated top-K approach, even though the exponential size of local flatlines does not affect the process, since local flatlines are avoided, improvement in accuracy is not guaranteed as in the exhaustive top-K approach.

There are two cases that can be identified, which determine whether the accuracy improves or worsens when traversing the truncated subgraph. One case is when at the root of the subgraph most of the v pairs have been assigned the correct semantic relationship. In this case traversing the truncated subgraph worsens the currently achieved schema mapping accuracy. This case is identified when at the root of the subgraph $c_v > f_v$ and $Bel_v(\overset{S}{\approx}) > Bel_v(\{\overset{S}{\gamma}\})$ or $c_v < f_v$ and $Bel_v(\overset{S}{\approx}) < Bel_v(\{\overset{S}{\gamma}\})$, where $Bel_v(\overset{S}{\approx}))$ is the belief of compatibility for all v pairs and $Bel_v(\{\overset{S}{\gamma}\})$ is the

belief of incompatibility for the v pairs.

The second case is when at the root of the subgraph most of v pairs have not been assigned the correct relationships. This case is identified when at the root of the subgraph we have that $c_v < f_v$ and $Bel_v(\overset{S}{\sim}) > Bel_v(\{\overset{S}{\gamma}\})$ or $c_v > f_v$ and $Bel_v(\overset{S}{\sim}) < Bel_v(\{\overset{S}{\gamma}\})$ In this case traversing the truncated subgraph improves the current achieved schema mapping accuracy.

Proof. The root of each subgraph specified by the v pairs of objects in the exhaustive top-K tree represents exactly the same mapping as the root in the truncated version of the sub-graph.

- $Bel_v(\overset{S}{\sim}) > Bel_v(\overset{S}{\gamma})$ Assuming that compatibility has higher belief than incompatibility for the v pairs, $Bel_v(\overset{S}{\sim}) > Bel_v(\overset{S}{\gamma})$, the root of the truncated sub-graph assigns to all v pairs the compatibility relationship. The accuracy A_0 at the root is the same as in Equation 5.8:

$$\begin{aligned} A_0 &= 1 - \frac{(f_{0v} + f_{0(N-v)}) + (m - (c_{0v} + c_{0(N-v)}))}{m} \\ &= 1 - \frac{m - c_{0(N-v)} + f_{0(N-v)}}{m} - \frac{f_{0v} - c_{0v}}{m} \end{aligned} \tag{5.9}$$

  where c_{0v},f_{0v} are the true and false positives, respectively, regarding the v pairs of objects at the root of the subgraph with $c_{0v} + f_{0v} = v$. $c_{0(N-v)}$,$f_{(N-v)}$ are the true and false positives, respectively, regarding the $N - v$ pairs at the root.

  At the next node of the truncated sub-graph, the v pairs are configured to be incompatible, which is the relationship with next highest belief for these pairs. Therefore, the true and false positives in the v pairs are $c_{1v} = f_{1v} = 0$. Since the configuration of the $N - v$ pairs does not change in this node, we have $c_{1(N-v)} = c_{0(N-v)}$ and $f_{1(N-v)} = f_{0(N-v)}$. The accuracy A_1 at this node is:

$$
\begin{aligned}
A_1 &= 1 - \frac{m - c_{1(N-v)} + f_{1(N-v)}}{m} - \frac{f_{1v} - c_{1v}}{m} \\
&= 1 - \frac{m - c_{0(N-v)} + f_{0(N-v)}}{m} - 0 \\
A_1 &= A_0 + \frac{f_{0v} - c_{0v}}{m} \qquad (5.10)
\end{aligned}
$$

Based on Equation (5.10), there are two cases:

1. $f_{0v} < c_{0v}$: $(f_{0v} - c_{0v})/m < 0$, which means that $A_1 < A_0$
2. $f_{0v} > c_{0v}$: $(f_{0v} - c_{0v})/m > 0$, which means that $A_1 > A_0$

The first case describes the situation where there are more true positives (c_{0v}) pairs than false positives (f_{0v}) in the v pairs at the root of the subgraph, *i.e.* most of the semantic mappings for the v pairs have been correctly identified. Traversing the truncated top-K tree in this case worsens the accuracy of the schema mappings; the root has higher belief A_0 than the leaf A_1.

The second case where $f_{0v} > c_{0v}$, describes the situation where there are more incompatible pairs than compatible, which does not agree with the tool's beliefs. The truncated top-K approach in this case guarantees improvement in accuracy; the leaf has higher belief than the root, $A_1 > A_0$.

- $Bel_v(\overset{S}{\sim}) < Bel_v(\overset{S}{\not\sim})$ Assuming that incompatibility has higher belief than compatibility in the v pairs, the accuracy at the root of the truncated sub-graph is $A_0 = 1 - \frac{m - f_{0(N-v)} - c_{0(N-v)}}{m}$, while the accuracy at the leaf is $A_1 = 1 - \frac{m - c_{1(N-v)} + f_{1(N-v)}}{m} - \frac{f_{1v} - c_{1v}}{m}$.

  Similar to the previous case, we can deduce that $A_1 = A_0 - \frac{f_{1v} - c_{1v}}{m}$. Now, if $c_{1v} > f_{1v}$ then, $A_1 > A_0$, but if $c_{1v} < f_{1v}$ then $A_1 < A_0$.

□

We could assume that most of the v pairs are correctly identified when the tool has been trained extensively. Thus, the exhaustive top-K approach could be applied after extensive training, while the truncated top-K approach could give better results in the early stages of training. Additionally, the truncated top-K approach could be useful when dealing with matching tasks that do not follow the general patterns of other matching tasks. In these cases, extensive training is not helpful.

5.3.5 Using regression

In Section 5.1.3, we showed that our USMs are produced based on a training table, which is divided based on degree ranges. Additionally, we showed that by using regression on the training table's data we can possibly produce more precise USMs, *e.g.* in ranges where there are no training data the trainer does not need to show total ignorance but it can derive a USM based on the USMs on the adjacent ranges. In this section, we are going to show that our experiments confirm our intuition, even though the improvement gained from using regression is small.

We compared the accuracy of the top-1 schema mapping produced by the three matching experts, Similarity Flooding, Rondo String Matching and WordNet, both with and without using regression to derive the USMs. For each user and each task we identified the maximum accuracy of the experts in both cases, and we identified the percentage of improvement in accuracy when using regression. For example, assume that for each user u and for each task i expert j produces maximum accuracy a_{reg}^{uij} and a^{uij} with and without regression respectively. Improvement in accuracy is calculated as the fraction of the difference between the two accuracies, $a_{reg}^{uij} - a^{uij}$, over the maximum possible difference, $1 - a^{uij}$:

$$Impr_{uij} = \frac{a_{reg}^{uij} - a^{uij}}{1 - a^{uij}} \tag{5.11}$$

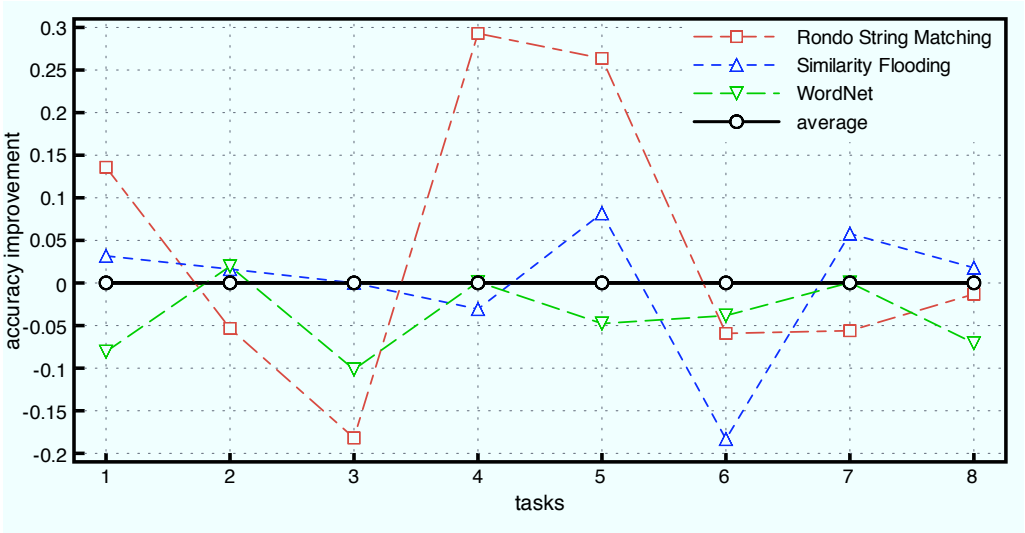


Figure 5.8: Accuracy improvement when using a regression trainer

$Impr_{uij}$ equals 1 when the maximum possible improvement is achieved, and $Impr_{uij}$ is negative if the accuracy when using regression is worse than the accuracy when regression is not used.

Figure 5.8 shows the average accuracy improvement $Impr_{ij}$ across all users for each task i and each expert j in top-1 matching. As seen, using regression does not guarantee an improvement in accuracy, but a small increase is achieved. On average the three experts show an improvement of 0.0003. More particularly, the Similarity Flooding expert shows an improvement of -0.001 on average across all tasks, the Rondo String Matching expert shows an improvement of 0.041 and the WordNet experts shows a -0.039 improvement.

5.3.6 Types of Experiments

We have performed two types of experiments of our schema matching implementation. The experiments show that our top-K matching approach improves the accuracy of the matching algorithms and that the improvement in accuracy gained from the top-K schema mappings outweighs the user *selection cost* of going through

the mappings to identify the most preferable one. In the experiments described in the rest of this chapter only regression-based experts have been used.

In the first set of experiments, we want to examine whether our translation of similarity degrees to USMs degrades the accuracy of the schema matching algorithms. To achieve this, we compare the accuracy of the single schema mapping produced by each expert when similarity degrees are produced, against the accuracy of the top-1 schema mapping produced by the same expert when it is adapted to translate similarity degrees to USMs.

In the second set of experiments, we use our matching tool to compare the improvement in the accuracy when top-K schema mappings are identified against the accuracy of the top-1 schema mapping. However, the production of top-K schema mappings entails a user cost of selecting the most preferable mapping out of the top-K. Therefore, in these experiments we also take into consideration the maximum cost that the user would have to pay to interactively go through the top-K schema mappings to identify a preferable mapping.

Regarding the first set of experiments, the single schema mapping identified by traditional matching, *i.e.* where uncertainty is not considered, needs to be computed based solely on the similarity degrees. To identify the single schema mapping, we used the filtering method introduced in [73] as *perfectionist egalitarian polygamy*, and mentioned in [33] as *MaxDelta* with relative tolerance value $d = 1$. This filtering method produces schema mappings with the highest accuracy in [73]. The idea behind this filtering method is that each schema object s_1 is matched only to the schema object s_2 with the highest similarity degree and vice versa, *i.e.* s_1 matches with s_2 if and only if for all x, $sim(s_1, s_2) \geq sim(s_1, s_x)$ and $sim(s_1, s_2) \geq sim(s_x, s_2)$.

5.3.7 Experimental Evaluation of Top-1

In our first set of experiments, we want to evaluate the accuracy of our tool that outputs USMs instead of similarity degrees.

We remind the reader that our data set contains eight matching tasks. For each task seven different users have manually specified their desired schema mapping. Each *run* of our experiments is executed for a specific user u, a specific target task t and a specific training set size. In each *run* the number of individual matching experiments depend on the training set size. Table 5.2 lists the number of experiments executed for each training set size. The result of each *run* is the average accuracy of the top-1 schema mappings identified in the individual experiments of the *run*.

For each user u and each target task t, seven *runs* are executed, which correspond to 127 experiments. Since there are eight matching tasks, for each user u we have executed $8 \times 127 = 1016$ individual experiments. Since there are seven users, the total individual matching experiments we have executed in this first set of experiments is $7 \times 1016 = 7112$ experiments.

First of all, we want to confirm that by increasing the training set size the tool performs better, *i.e.* that the average accuracy of the top-1 schema mapping improves. Figure 5.9 illustrates the average accuracy of each expert as the training set size increases. The horizontal axis specifies the training set size and the vertical axis specifies the average accuracy of the expert across all users and all tasks. For example, the average accuracy of the Similarity Flooding expert across all seven users and all eight tasks is -0.0877 when the training set size is equal to one. As we can see from the figure, in general the average accuracy of each expert improves as the training set size increases.

Figure 5.10(a) illustrates the best accuracy achieved on average across all users for each task by each expert. Figure 5.10(b) illustrates how this best accuracy compares

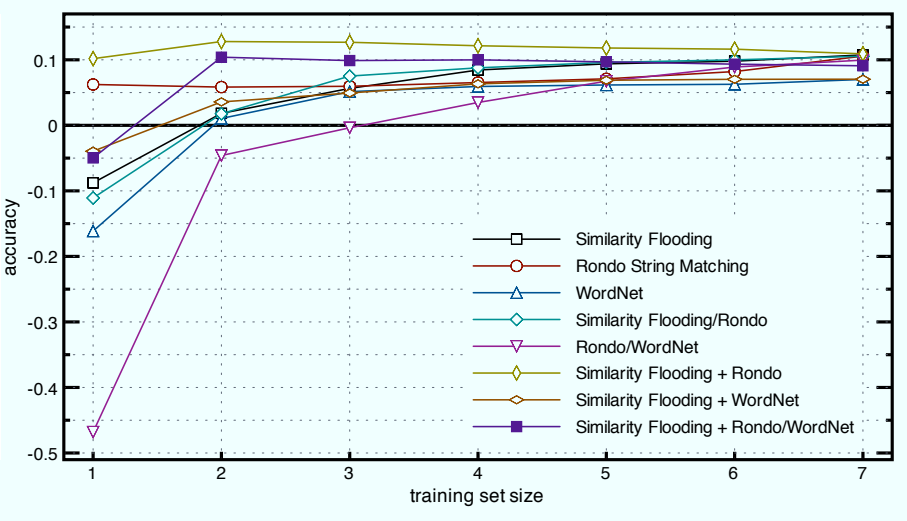
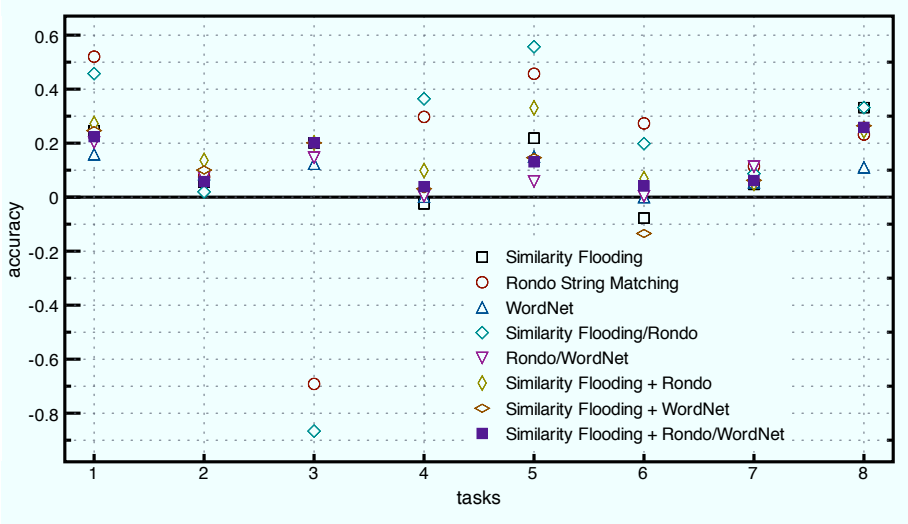


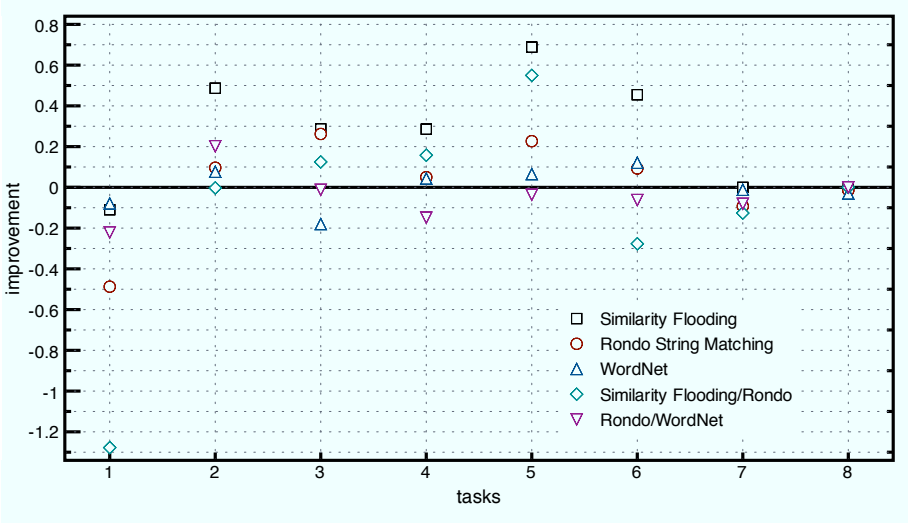
Figure 5.9: Average accuracy of Top-1 schema mapping as training set size increases

with the best accuracy achieved when using the perfectionist egalitarian polygamy approach. The formula used for this comparison is similar to Equation 5.11, *i.e.* it identifies the ratio of the actual improvement achieved compared to the maximum possible improvement. If the accuracy produced by our tool is less than the accuracy produced by the polygamy approach, improvement is negative. Note that in Figure 5.10(b) we compare only the results of singleton experts since we want to compare how the same algorithms perform in our approach and in the perfectionist egalitarian polygamy approach.

As we can see in Figure 5.10(b), in most tasks using our approach improves the accuracy of an algorithm. The interesting cases are task 1 and task 3. In task 1, even though our experts achieve a high accuracy, *e.g.* the Similarity Flooding/Rondo expert achieves an accuracy of 0.457, this is still worse than the accuracy achieved by the polygamy approach. For example, the accuracy of the Similarity Flooding/Rondo expert is -1.27 worse than if the polygamy approach was used. Thus, we see that the polygamy approach performs exceptionally well for task 1. On the other hand, on task 3, our tool in some cases produces very low accuracy, *e.g.* the Similarity Flooding/Rondo expert produces accuracy -0.86. However, it still performs better than the polygamy approach, *e.g.* the Similarity Flooding/Rondo expert improves



(a) Average best accuracy



(b) Accuracy improvement

Figure 5.10: Average best accuracy across all users and training set sizes for Top-1 schema mapping

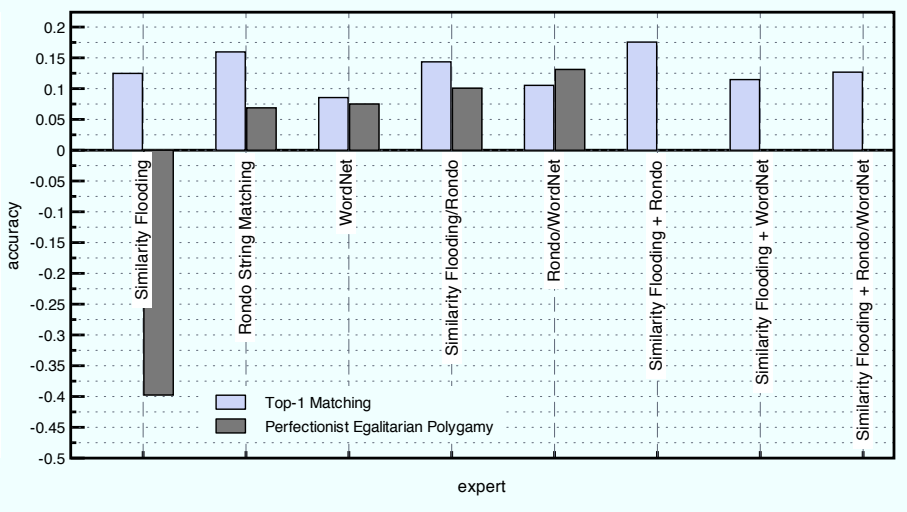


Figure 5.11: Average best accuracy by each expert across all tasks and all users for Top-1 schema mapping

the accuracy of task 3 by 0.1247.

Therefore, we could say that for the same matching algorithms our approach produces more consistent accuracy results:

- it produces an accuracy which does not match the polygamy accuracy but it is still fairly high (task 1),
- it improves in general the polygamy accuracy (tasks 2-6),
- it matches the polygamy accuracy (tasks 7 and 8).

Figure 5.11 illustrates the average best accuracy achieved by our approach and the perfectionist egalitarian polygamy approach across all tasks and all users for each expert. We see that our approach improves the accuracy of all experts, except from the Rondo/WordNet expert. More importantly, we see that the most accurate set up (accuracy of 0.1755) is using USMs and combining the Similarity Flooding and Rondo String Matching experts. This set up is more accurate even than the hybrid Similarity Flooding/Rondo expert (accuracy of 0.1434).

5.3.8 Experimental Evaluation of Top-K

In our second set of experiments, we compare the accuracy of the top-3 schema mappings against the top-1 schema mapping. We want to confirm our intuition that by considering further schema mappings and not just a single mapping, as most current approaches do, we achieve better accuracy.

As in the evaluation of the top-1 matching, for each user u and each target task t, seven *runs* are executed. In total, the same number of individual matching experiments are executed, *i.e.* 7112 experiments, but now the top-3 schema mappings are computed, instead of just the top-1. The exhaustive top-K approach is used to compute the top-3 match results. In each experiment, the most accurate schema mapping out of the top-3 is identified. The results of each experiment are: (a) the maximum improvement of accuracy gained in the top-3 mappings compared to the accuracy of the top-1 mapping, and (b) the worst user cost needed to identify the most preferable mapping in the top-3. The result of each *run* is the average maximum improvement of accuracy and the average worst user cost.

First, we want to confirm that by increasing the training set size the average maximum accuracy of the top-3 schema mappings improves. Figure 5.12 illustrates the average maximum accuracy for each expert as the training set size increases. As we can see from the figure, accuracy improves for the exhaustive top-3 approach as training set size increases.

To confirm our previous intuition in Section 5.3.4 that the exhaustive top-K approach provides better results after extensive training, while the truncated top-K approach is useful in the early stages of training, we have also examined how the average maximum accuracy of the truncated top-3 compares to the one of the exhaustive top-3. We compared the two approaches in similar way to Section 5.3.5, *i.e.* we calculate the ratio of the improvement gained compared to the maximum possible improvement that could be gained. The results are illustrated in Figure 5.13. The

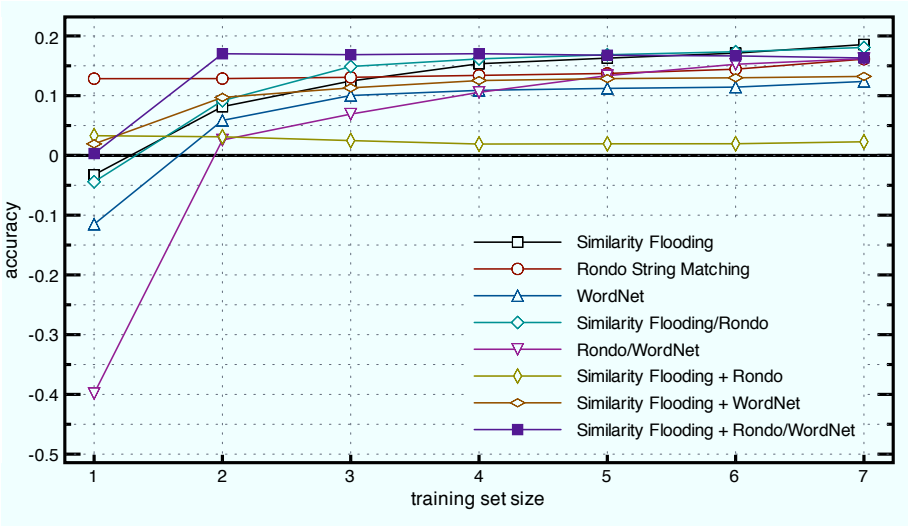


Figure 5.12: Average maximum accuracy of exhaustive Top-3 schema mappings as training set size increases

figure shows that, in general, as the training set size increases the improvement gained in the truncated top-3 approach decreases compared to the exhaustive top-3 approach.

Going back to the exhaustive top-3 experiments, Figure 5.14(a) illustrates the improvement in accuracy when looking at the top-3 schema mappings compared to the accuracy of the top-1 mapping. In the figure, the improvement in top-3 is illustrated against the training set size. First of all, we see that the top-3 approach always improves against the top-1, *i.e.* our tool correctly identifies the pairs of objects whose semantic mapping is incorrect in the top-1 mapping and repairs them. As we were expecting, in the exhaustive top-3 approach the improvement is not gigantic, but it is substantial, *e.g.* above 4% in all cases. The reason is that in the exhaustive top-3 schema mappings, the semantic mappings of a maximum of two pairs of objects can be changed compared to the top-1 schema mapping.

The aforementioned improvement in accuracy does not come for free. The user has to manually go through the top-3 mappings and choose the best one, *i.e.* a selection cost must be paid. Figure 5.14(b) illustrates the ratio of the improvement against the worst selection cost. For each individual matching experiment, worst selection

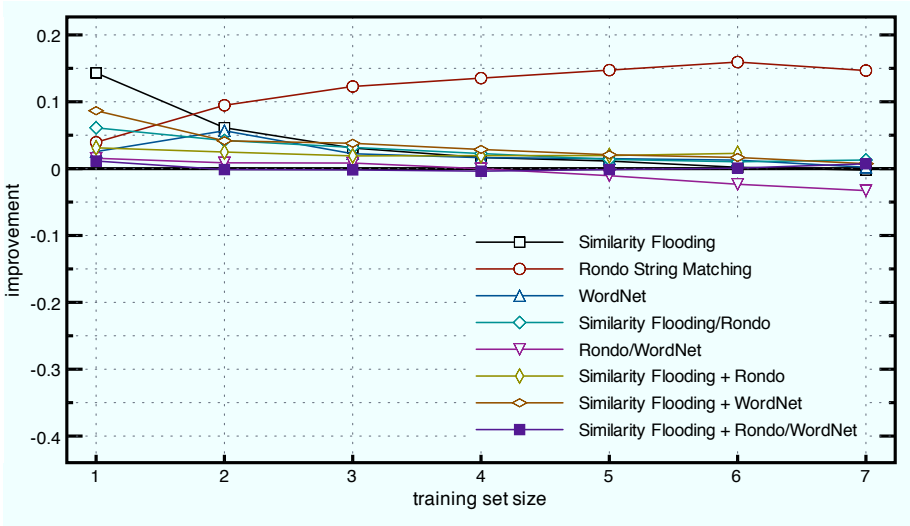
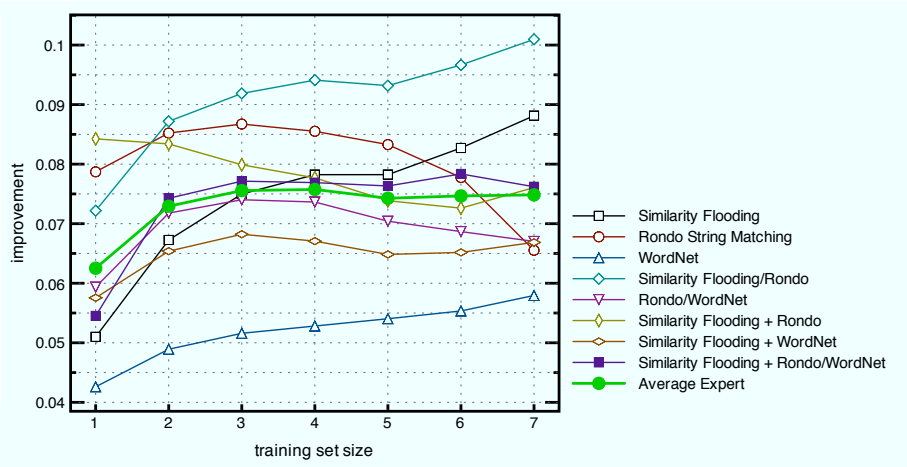


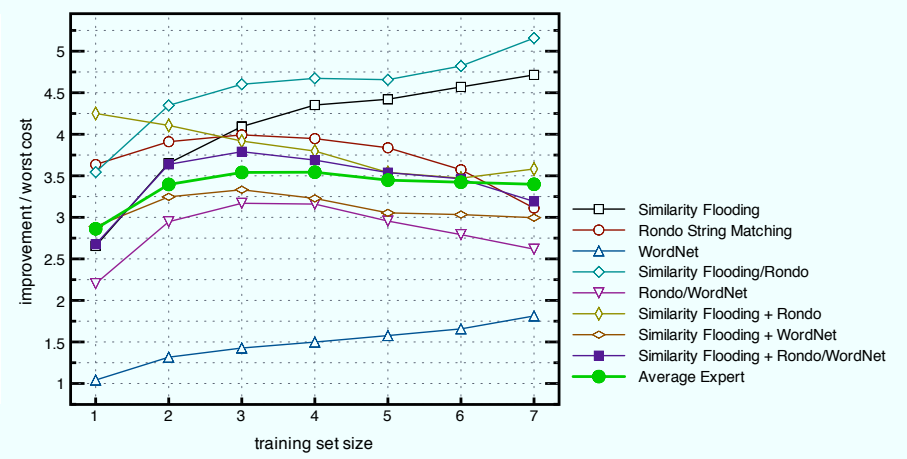
Figure 5.13: Average maximum improvement of accuracy of the truncated top-3 schema mappings against the exhaustive Top-3 schema mappings as training set size increases

cost is calculated as the ratio of the number of pairs whose semantic mapping has been changed in the top-3 schema mappings over the total number of pairs in the matching task, as shown in Example 5.4. As we can see from the figure, on average the improvement in accuracy gained is three times more than the cost needed to obtain it. Thus, our approach is cost efficient. Note that in the truncated top-K approach we do not need to examine the selection cost, since this approach is useful only in a fully automatic setting.

Figure 5.15 illustrates the best accuracy achieved on average across all users for each task by each expert when computing the top-3 schema mappings. The comparison of the top-3 accuracy against the top-1 accuracy for each task is shown in Figure 5.16. As we can see, the WordNet expert is the only expert which does not improve in the top-3 mappings for all tasks. The Rondo String Matching expert improves all tasks showing its least improvement 0.018 for task 5 and its maximum improvement 0.115 for task 8. The Similarity Flooding + WordNet expert improves all tasks, with least improvement 0.0038 for task 3 and maximum improvement 0.129 for task 1. In general, we can see that the experts by looking at the top-3 schema mappings



(a) Improvement



(b) Improvement over cost

Figure 5.14: Top-3 matching accuracy improvement against Top-1 and cost

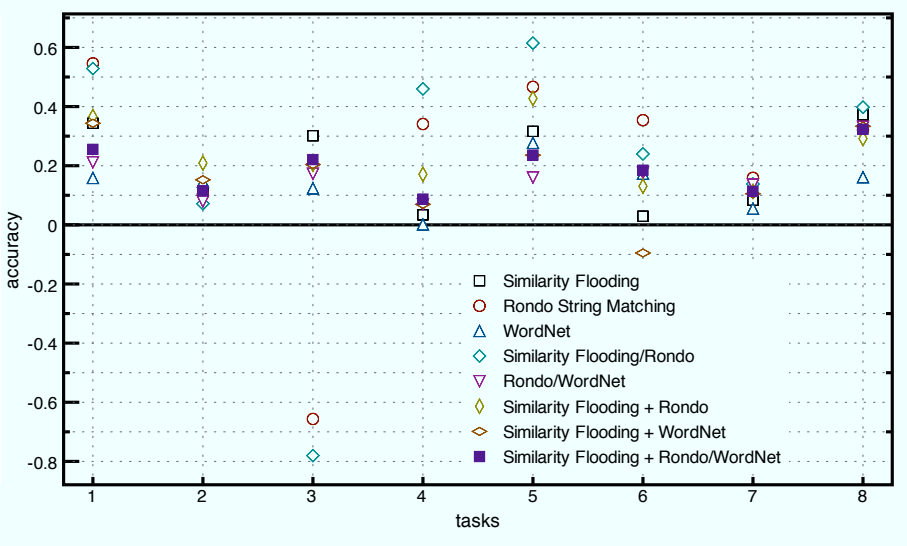


Figure 5.15: Average best accuracy across all users and training set sizes for top-3 schema mapping

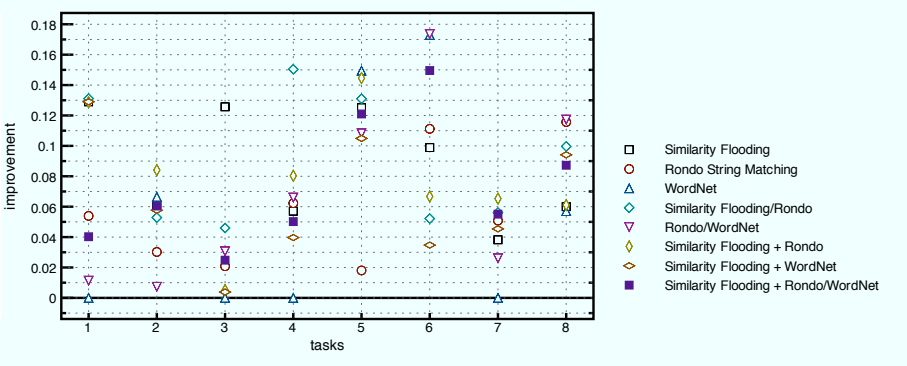


Figure 5.16: Accuracy improvement by each expert across all tasks and all users for Top-3 schema mapping against Top-1 mapping

improve their accuracy compared to just looking the top-1 mapping.

Finally, Figure 5.17 illustrates the average best accuracy achieved in the top-3 and the top-1 approaches across all tasks and all users for each expert. We see that the experts further improve the accuracy of the top-1 mapping. In particular, the Similarity Flooding + Rondo expert shows the highest overall accuracy of 0.24, which is the highest achieved by any expert in top-3, top-1 and the perfectionist egalitarian polygamy approaches.

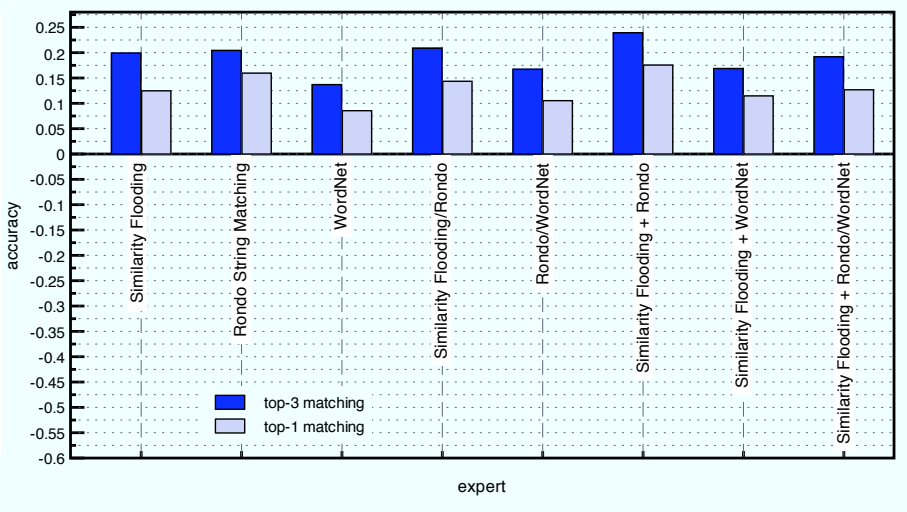


Figure 5.17: Average best accuracy by each expert across all tasks and all users for Top-3 schema mapping

5.4 Summary

In this chapter, we have presented our implementation of schema matching.

In our schema matching process, we produce uncertain semantic mappings (USMs), which allow the representation of the uncertainty of the automatic schema matching tool. USMs require the distribution of probability masses to the several semantic relationships possible for each pair of objects. Thus, in this chapter we explain a methodology for the identification of these probability masses.

Our matching tool is based on existing matching algorithms. These algorithms produce similarity degrees for each pair of objects, which need to be translated into probability masses. To achieve this, the matching tool is trained based on user-validated semantic mappings from previous matching tasks and stores statistical information about these mappings. This information can then be used to identify probabilities for sets of semantic relationships. Compared to existing approaches [46, 81] that could potentially support the representation of uncertainty for multiple semantic mappings, our implementation relies on the user-validated semantic mappings. In [81], the identification of the probability mass is based on training

data produced by matching identical schemas. When matching identical schemas, an object in one schema is equivalent to itself in the other schema. However, this process can only be used to derive equivalence relationships between schema objects. Additionally, the final result of matching would be identical for any user. In our implementation, the final result of matching is user-specific, which is closer to reality where different users have different views on how schemas should be matched [73]. In [46], schema matching uncertainty is represented based on fuzzy set theory. In fuzzy set theory (see Section 4.6), instead of probabilities, *grades* of membership must be identified. In [46], these grades of membership are the similarity degrees produced by existing matching algorithms like the ones we have used for our experts. Additionally, [46] (and [81]) does not directly support the representation of uncertainty on multiple semantic mappings, but its implementation needs to be extended. We provide a methodology directly applicable on multiple semantic mappings.

The produced USMs specify an exponential number of possible schema mappings for each matching task. To deal with this complexity problem, our tool identifies the top-K most certain schema mappings. We have implemented two approaches for identifying the top-K mappings: the exhaustive and the truncated approach. In this chapter, we show: (a) that the exhaustive top-K approach guarantees a more accurate schema mapping compared to the top-1 mapping, and (b) that the exhaustive top-K approach might be slow to identify a good schema mapping in an environment with no user interaction. The truncated top-K approach is shown to be more effective when there is no user interaction and when the matching tool has not been extensively trained.

In our experimental evaluation of our implementation, we show that our approach improves the accuracy of existing matching algorithms and that the exhaustive top-K matching further improves these results in a cost effective manner.

Chapter 6

Schema Merging

In the previous chapter, we discussed in detail the identification of the **top-K schema mappings** in our matching approach. As explained in Section 4.8, these K schema mappings define K schema merging tasks. These tasks produce the **top-K integrated schemas** (Definition 4.12). In this chapter, we are going to introduce our schema merging process and show how each integrated schema is produced.

Our approach to schema merging uses formal and precise low-level transformation rules. One of the advantages of our merging approach, which is due to the formal definition of our rules, is that we are able to prove the **soundness** and **completeness** of the rules; our rules do not produce any **information loss** nor **gain**. Additionally, the precision in which these rules are defined allows us to examine the degree to which these rules can be automated.

Compared to existing approaches [59, 101, 89] that consider similar semantic mappings to our approach, our schema merging also specifies view definitions between the input and the integrated schemas. Compared to the merging approaches [87, 74, 12], these approaches consider data mappings, whose discovery is a very hard problem [31], and in addition these approaches are model specific. Instead, our schema merging approach specifies merging rules based on semantic mappings. Our merging rules

are defined using a low-level data model, the HDM (presented in Chapter 3), which allows their translation to higher-level rules for multiple high-level model schema merging [94].

The structure of this chapter is as follows. Section 6.1 presents our low-level schema merging approach, discussing the possible merging rules (Sections 6.1.1-6.1.4), their properties (Section 6.1.5) and the degree to which they can be automated (Section 6.1.6). Section 6.2 explains our generic schema merging framework and Section 6.3 provides our methodology for defining merging rules for high-level models based on the generic framework. As an example, we illustrate how generic rules can be translated into ER data model rules. Finally, Section 6.4 gives an example of the final outcome of our top-K schema merging.

6.1 Low-Level Schema Merging

We have defined our low-level schema merging methodology based on a set of formally defined rules. These rules have been identified by exhaustively investigating the semantic mappings between any pairs of objects and considering all possible configurations of sub-schema structures.

The input to our schema merging process is a schema mapping (Definition 4.6), which is a list of N semantic mappings. For each one of these semantic mappings, we identify which merging rule is applicable. The application of all the identified rules produces the final integrated schema. Note that our rules have been defined with the assumption that the data in the input schemas have been pre-processed and are conformed. Therefore, our merging rules do not use any instance translation functions.

In order to avoid the complexity of high-level models, we have used the HDM (Chapter 3) to define our rules. The HDM contains two existential constructs: nodes and

edges. Thus, to identify all possible cases of semantic mappings between pairs of objects, we only have to consider each possible semantic relationship between two nodes and each possible semantic relationship between two edges.

For each identified schema merging case, either (i) a simplification is possible and a rule is specified, (ii) a simplification is not possible (NSP), or (iii) the case cannot logically exist and it is identified as an SRNP (Semantic Relationships Not Possible) case.

Based on the identified cases and their respective rules, we see that we can apply a standard schema integration approach [8], where integration is split into three phases: (a) naming conforming, (b) unioning and (c) restructuring. In our schema merging approach, each one of these phases is associated with a specific set of rules.

6.1.1 Summary of our Schema Merging rules

Table 6.1 provides a summary of all the rules used in our schema merging approach.

The first column, *case*, of the table specifies the sub-schema structures that are examined in the table. The first row of the table examines the semantic relationship x between any two nodes A and B. The rest of the rows of the table examine each possible semantic relationship x between two edges.

In the first row, all possible semantic relationships between two nodes are investigated: equivalence ($A \overset{S}{\equiv} B$), subsumption ($A \overset{S}{\subset} B$), intersection ($A \overset{S}{\cap} B$) and disjointness ($A \overset{S}{\not\cap} B$). For each case, a rule is specified. Incompatibility ($\overset{S}{\not\sim}$) does not need to be examined, since by default an incompatibility mapping implies that no schema changes are necessary.

In the second, third and fourth row, all possible semantic relationships between two edges are investigated. Each row identifies a different sub-schema configuration of the two edges. The second row of the table identifies the case where the two

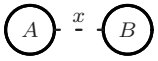
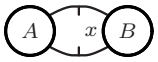
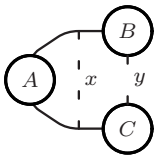
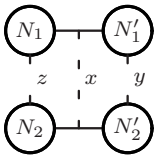
case	y	z	x: $\subseteq_S$	x: $\subset_S$	x: $\supset_S$	x: $\not\subset_S$
			Node Merge Node Distinction	Addition of Inclusion	Addition of Intersection	Addition of Union
			Edge Merge Edge Distinction	NSP	Addition of Edge Intersection	NSP
	$\subset_S$		Redundant Edge Removal	Optional Edge Removal	Addition of Edge Intersection	NSP
	$\supset_S$		Specialization of Edges	NSP	Addition of Edge Intersection	NSP
	$\not\subset_S$		SRNP	SRNP	SRNP	Generalization of Edges
	$\subseteq_S$	$\subset_S$	Redundant Edge Removal	Optional Edge Removal	Addition of Edge Intersection	NSP
		$\supset_S$	Specialization of Edges	Optional Edge Removal	Addition of Edge Intersection	NSP
		$\not\subset_S$	SRNP	SRNP	SRNP	Generalization of Edges
		$\not\supset_S$	Redundant Edge Removal	NSP	Addition of Edge Intersection	NSP
	$\supset_S$	$\subset_S$	Specialization of Edges	Optional Edge Removal	Addition of Edge Intersection	NSP
		$\supset_S$	Specialization of Edges	NSP	Addition of Edge Intersection	NSP
		$\not\subset_S$	SRNP	SRNP	SRNP	Generalization of Edges
	$\not\subset_S$	$\subset_S$	SRNP	SRNP	SRNP	Generalization of Edges
		$\supset_S$	SRNP	SRNP	SRNP	Generalization of Edges
		$\not\subset_S$	SRNP	SRNP	SRNP	Generalization of Edges

Table 6.1: Summary of all merging rules possible for all combinations of the semantic mappings

edges associate the same two nodes A and B. The third row of the table identifies the case where the two edges have one node A in common. This row apart from examining each possible semantic relationship x between the edges, in addition examines each possible semantic relationship y between the nodes B and C that node A is associated with by the two edges. Note here that the equivalence ($\stackrel{\mathrm{S}}{=}$) relationship is not considered for y, since this case would be semantically identical with the configuration of the second row of the table. The final row of the table identifies the case where the two edges do not have any node in common. In addition to examining all possible relationships for x, also the relationships y and z between the nodes at the ends of the two edges are examined. Again, the equivalence relationship is not considered for either y or z, since it would imply the configuration of either the second or third row of the table.

While exhaustively examining all possible cases of semantic relationships, cases where the semantic relationships cannot co-exist have been identified. For example, consider in the third row of Table 6.1 the case where the semantic relationship x between the two edges is equivalence and the semantic relationship y between the nodes B and C is disjointness. This is an illegal case, since by definition the disjoint nodes B and C do not have any common instances, therefore the two edges cannot have any common instances either. Thus, the edges cannot be equivalent. This is an illegal case and it is identified in the Table 6.1 with the acronym SRNP (Semantic Relationships Not Possible).

Additionally, we have identified cases where the schema cannot be simplified based on the specified semantic relationships. These cases are illustrated as NSP (No Simplification Possible) cells in Table 6.1, and no rules have been defined for them. For example, edge disjointness cases can only be simplified if the nodes in at least one end of the edges are disjoint. As it will be explained in Section 6.1.5, if this condition holds then the application of the corresponding rule produces a complete and sound schema with no additional or missing instances, *i.e.* no *information loss*

or *gain*. In general, we require all our merging rules not to cause any information loss nor gain. More about this requirement, which we call **Intended Domain Preservation Property** (**IDPP**), is discussed in Section 6.1.5.

6.1.2 Naming Conforming

Based on a standard schema integration approach [8], the first phase of merging is naming conforming.

In the naming conforming phase, naming conflicts between the objects of the schemas are resolved. Given that there may be both homonym and synonym [8] conflicts between nodes and between edges, this leads us to have four rules: **Node Merge**, **Edge Merge**, **Node Distinction** and **Edge Distinction** rules. The Node Merge and Edge Merge rules resolve the synonym conflict of having two equivalent nodes or two equivalent edges that do not have identical names. In such a case, the rules assign to the two equivalent objects a common name. The Node Distinction and Edge Distinction rules resolve the homonym conflict of having two non-equivalent objects with identical names. In this case, the objects have to be assigned distinct names to make them distinguishable. Thus only the equivalence semantic relationship is used in this phase of schema merging.

Based on the above observations, the following auxiliary functions are required:

- `identicalNames`(N_1, N_2): checks whether two names N_1 and N_2 are identical. Returns $true$ when the two names N_1, N_2 are identical, and $false$ otherwise.
- `commonName`(N_1, N_2, N): supplies a common name N based on names N_1 and N_2. Variables N_1 and N_2 must be grounded.
- $\neg\ \langle\langle\langle N_1\rangle\rangle, rel_i, \langle\langle N_2\rangle\rangle\rangle$: returns $true$ if the mapping of the objects $\langle\langle N_1\rangle\rangle$ and $\langle\langle N_2\rangle\rangle$ is $\langle\langle\langle N_1\rangle\rangle, rel_j, \langle\langle N_2\rangle\rangle\rangle$ with $rel_j \in \Theta_{rel} = \{\overset{S}{=}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\cap}, \overset{S}{\not\supset}\}$ and $rel_i \neq rel_j$, and $false$ otherwise.

$$\frac{\langle\langle N_1\rangle\rangle, \stackrel{S}{=}, \langle\langle N_2\rangle\rangle \rangle \quad \neg\ \texttt{identicalNames}(N_1, N_2) \quad \texttt{commonName}(N_1, N_2, N')}{\mathsf{renameNode}(\langle\langle N_1\rangle\rangle, \langle\langle N'\rangle\rangle) \quad \mathsf{renameNode}(\langle\langle N_2\rangle\rangle, \langle\langle N'\rangle\rangle)}$$

(a) Node Merge

$$\frac{\neg\ \langle\langle N_1\rangle\rangle, \stackrel{S}{=}, \langle\langle N_2\rangle\rangle \rangle \quad \texttt{identicalNames}(N_1, N_2) \quad \texttt{distinctNames}(N_1, N_2, N_1', N_2')}{\mathsf{renameNode}(\langle\langle N_1\rangle\rangle, \langle\langle N_1'\rangle\rangle) \quad \mathsf{renameNode}(\langle\langle N_2\rangle\rangle, \langle\langle N_2'\rangle\rangle)}$$

(b) Node Distinction

$$\frac{\langle\langle e_1, N_1, N_2\rangle\rangle \stackrel{S}{=} \langle\langle e_2, N_1, N_2\rangle\rangle \quad \neg\ \texttt{identicalNames}(e_1, e_2) \quad \texttt{commonName}(e_1, e_2, e')}{\mathsf{renameEdge}(\langle\langle e_1, N_1, X_1\rangle\rangle, \langle\langle e', N_1, X_1\rangle\rangle) \quad \mathsf{renameEdge}(\langle\langle e_2, N_2, X_2\rangle\rangle, \langle\langle e', N_2, X_2\rangle\rangle)}$$

(c) Edge Merge

$$\frac{\neg\ \langle\langle e_1, N_1, N_2\rangle\rangle \stackrel{S}{=} \langle\langle e_2, N_1, N_2\rangle\rangle \quad \texttt{identicalNames}(e_1, e_2) \quad \texttt{distinctNames}(e_1, e_2, e_1', e_2')}{\mathsf{renameEdge}(\langle\langle e_1, N_1, N_2\rangle\rangle, \langle\langle e_1', N_1, N_2\rangle\rangle) \quad \mathsf{renameEdge}(\langle\langle e_2, N_1, N_2\rangle\rangle, \langle\langle e_2', N_1, N_2\rangle\rangle)}$$

(d) Edge Distinction

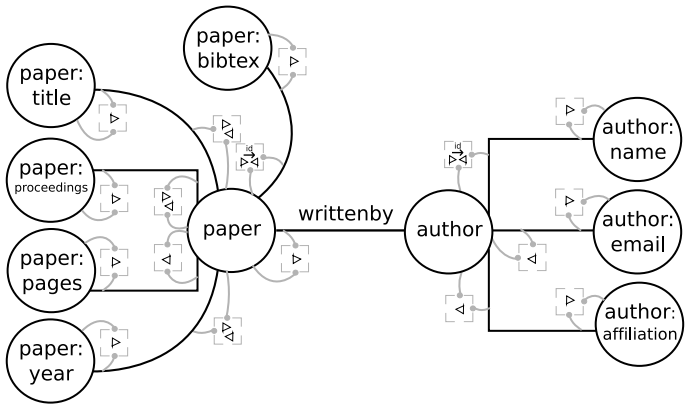
Figure 6.1: Naming Conforming rules

- $\texttt{distinctNames}(N_1, N_2, N_1', N_2')$: supplies with two distinct names N_1',N_2'. Variables N_1 and N_2 must be grounded and are used to derive N_1' and N_2' respectively.

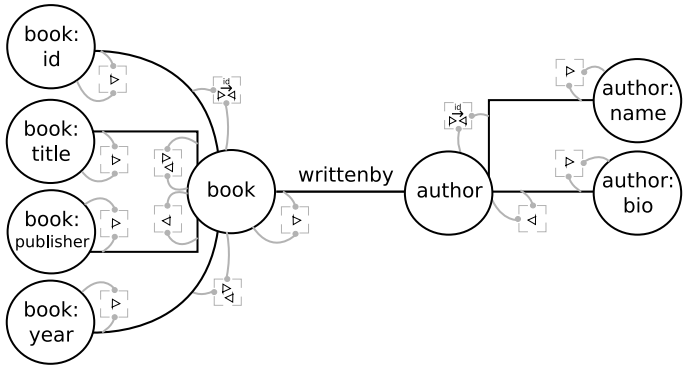
Figure 6.1 illustrates all the rules in the naming conforming stage. All our rules take the form of conditions, defined above the horizontal line, which if satisfied cause the BAV (see Section 2.3.2) transformations, below the line, to be generated.

For example, the Node Distinction rule has three conditions: the first checks that two nodes are not equivalent, the second checks that the names of the two objects are identical and the third condition identifies two distinct names. If these conditions are satisfied, then the two objects are renamed so that they have distinct names.

Notice also the Edge Distinction rule. By definition of the HDM (Definition 3.2),



(a) S_1



(b) S_2

Figure 6.2: HDM schemas S_1 and S_2

two edges e_1 and e_2 can have identical names as long they associate a different set of nodes/edges. For example, the edges $\langle\langle \mathsf{writtenBy}, \mathsf{book}, \mathsf{author} \rangle\rangle$ and $\langle\langle \mathsf{writtenBy}, \mathsf{paper}, \mathsf{author} \rangle\rangle$ can have identical names because one edge associates book and author while the other edge associates paper and author. Problems with the names of the edges arise if the edges associate the same nodes, *e.g.* if there are two writtenBy edges between book and author. In such a case, we need to identify whether the two edges represent the same real-world semantics or not, *i.e.* whether they are equivalent or not. The Edge Distinction rule only examines edges which associate the same nodes N_1 and N_2, they are not equivalent and finally have identical names. Only in these conditions we have two homonym edges that need to be renamed.

Example 6.1. Naming Conforming

In this chapter, we are going to use as a running example, the example presented in the introduction. In this Section 6.1 of the chapter, we are going to merge the HDM schemas that are produced from the ER schemas of the introduction. The HDM schemas, S_1 and S_2, are illustrated in Figure 6.2.

All compatibility mappings between S_1 and S_2 objects are listed below:

$\langle\langle\langle\langle \mathsf{paper} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \mathsf{book} \rangle\rangle\rangle$

$\langle\langle\langle\langle \mathsf{paper : bibtex} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \mathsf{book : id} \rangle\rangle\rangle$

$\langle\langle\langle\langle \_, \mathsf{paper}, \mathsf{paper : bibtex} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \_, \mathsf{book}, \mathsf{book : id} \rangle\rangle\rangle$

$\langle\langle\langle\langle \mathsf{paper : title} \rangle\rangle, \overset{S}{\cap}, \langle\langle \mathsf{book : title} \rangle\rangle\rangle$

$\langle\langle\langle\langle \_, \mathsf{paper}, \mathsf{paper : title} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \_, \mathsf{book}, \mathsf{book : title} \rangle\rangle\rangle$

$\langle\langle\langle\langle \mathsf{paper : year} \rangle\rangle, \overset{S}{\supset}, \langle\langle \mathsf{book : year} \rangle\rangle\rangle$

$\langle\langle\langle\langle \_, \mathsf{paper}, \mathsf{paper : year} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \_, \mathsf{book}, \mathsf{book : year} \rangle\rangle\rangle$

$\langle\langle\langle\langle \mathsf{writtenby}, \mathsf{paper}, \mathsf{author} \rangle\rangle, \overset{S}{\not\cap}, \langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{author} \rangle\rangle\rangle$

$\langle\langle\langle\langle \mathsf{author} \rangle\rangle, \overset{S}{\cap}, \langle\langle \mathsf{author} \rangle\rangle\rangle$

$\langle\langle\langle\langle \mathsf{author : name} \rangle\rangle, \overset{S}{\cap}, \langle\langle \mathsf{author : name} \rangle\rangle\rangle$

$\langle\langle\langle\langle \_, \mathsf{author}, \mathsf{author : name} \rangle\rangle, \overset{S}{\cap}, \langle\langle \_, \mathsf{author}, \mathsf{author : name} \rangle\rangle\rangle$

Based on the above mappings, the only rule that can be applied in the naming conforming phase is Node Distinction. The rule can be applied on the $\langle\langle \mathsf{author} \rangle\rangle$ nodes in S_1 and S_2 and on the $\langle\langle \mathsf{author : name} \rangle\rangle$ nodes.

We have that $\langle\langle\langle\langle \mathsf{author} \rangle\rangle, \overset{S}{\cap}, \langle\langle \mathsf{author} \rangle\rangle\rangle$, *i.e.* the two nodes are not equivalent: $\neg\langle\langle\langle\langle \mathsf{author} \rangle\rangle, \overset{S}{=}, \langle\langle \mathsf{author} \rangle\rangle\rangle$. In addition, the two nodes have identical names, therefore the call `identicalNames`(author,author) returns true. Thus, the first two conditions of the Node Distinction rule are satisfied.

The third condition calls the function `distinctNames`. There can be several automatic implementations of this function, *e.g.* the nodes can be prefixed

with the name of the node they are attached to:

distinctNames(author,author,paper_author,book_author).

The rule then generates the following transformations. The transformations are prefixed with the schema they should be executed on.

S_1.renameNode(⟨⟨author⟩⟩,⟨⟨paper_author⟩⟩)

S_2.renameNode(⟨⟨author⟩⟩,⟨⟨book_author⟩⟩)

Similarly, regarding the ⟨⟨author : name⟩⟩ nodes the following transformations should be executed:

S'_1.renameNode(⟨⟨author : name⟩⟩,⟨⟨paper_author : name⟩⟩)

S'_2.renameNode(⟨⟨author : name⟩⟩,⟨⟨book_author : name⟩⟩)

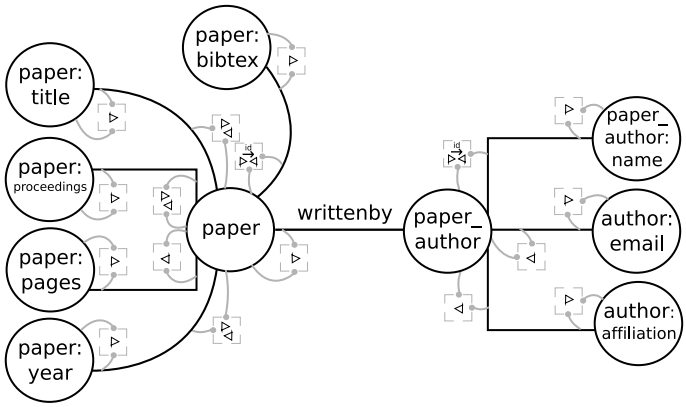
The resulting schemas S''_1 and S''_2 produced after the application of the Node Distinction rules are illustrated in Figure 6.3.

◇

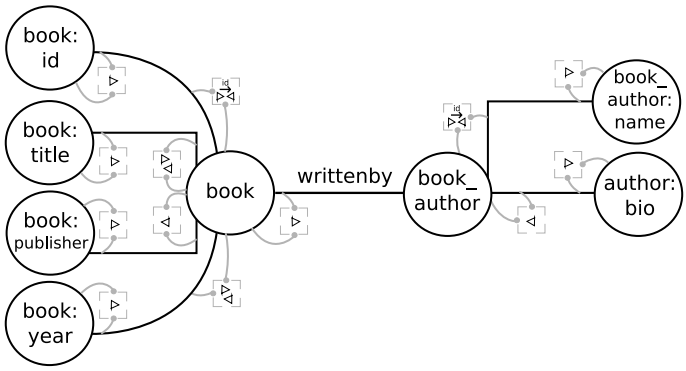
6.1.3 Unioning

The second phase of schema merging is the unioning phase [8]. In the unioning phase, the two schemas to be merged are superimposed; thus each pair of equivalent nodes/edges, which after the naming conforming stage have identical names, collapses into a single node/edge. The superimposition of schemas in BAV is performed by a series of extend transformations (Section 2.3.2). These transformations add to each schema any objects that are only available in the counterpart schema. Note that this is the only place where our merging approach uses extend, since it is stating here what is *not* directly available from one schema, but should only be sourced from the second schema.

In the rest of the unioning phase, subsumption, intersection and disjointness mappings between nodes are examined. The rules of this phase, illustrated in Figure 6.4, use the auxiliary function uniqueName(N_1,N_2,N',rel), which supplies with a new



(a) S_1''



(b) S_2''

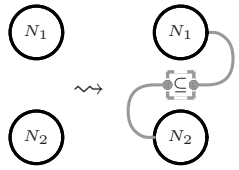
Figure 6.3: Schemas S_1'' and S_2'' produced after the naming conforming of S_1 and S_2 respectively

unique name N' based on the values of the grounded variables N_1 and N_2 and the semantic relationship *rel*. In detail, the rules of the unioning phase are:

- **Addition of Inclusion**: adds an inclusion constraint whenever a subsumption mapping exists between two nodes.
- **Addition of Intersection**: adds an *intersection* node, whenever an intersection mapping holds between two nodes. The *intersection* node added represents the common sub-domain of the intersecting nodes. In the rule (Figure 6.4(b)), the name of the new node $\langle\langle N' \rangle\rangle$ added is provided by the `uniqueName` function and its extent is defined as the instances that appear on both intersecting nodes $\langle\langle N_1 \rangle\rangle$ and $\langle\langle N_2 \rangle\rangle$. Additionally, two inclusion constraints are added to illustrate that the intersection node $\langle\langle N' \rangle\rangle$ is a subset of both $\langle\langle N_1 \rangle\rangle$ and $\langle\langle N_2 \rangle\rangle$.
- **Addition of Union**: adds a *union* node, whenever a disjointness mapping holds between two nodes. The *union* node added represents the union of the domains of the disjoint nodes. In the rule (Figure 6.4(c)), the extent of the new node $\langle\langle N' \rangle\rangle$ is defined by appending the instances of the disjoint nodes $\langle\langle N_1 \rangle\rangle$ and $\langle\langle N_2 \rangle\rangle$. Additionally, a union constraint is added to illustrate that $\langle\langle$N'$\rangle\rangle$ is the union of $\langle\langle N_1 \rangle\rangle$ and $\langle\langle N_2 \rangle\rangle$, and an exclusion constraint is added between $\langle\langle N_1 \rangle\rangle$ and $\langle\langle N_2 \rangle\rangle$ to illustrate that the disjoint nodes do not have any common instances.

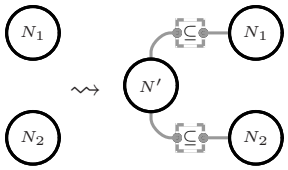
Example 6.2. Unioning In our running example, in the unioning phase first the schemas S_1'' and S_2'' are extended to produce a single schema where all the objects of both schemas appear. Then the Addition of Union rule on nodes $\langle\langle$paper$\rangle\rangle$ and $\langle\langle$book$\rangle\rangle$ is applied.

$$\frac{\langle \langle\!\langle N_2 \rangle\!\rangle, \overset{\mathrm{S}}{\subset}, \langle\!\langle N_1 \rangle\!\rangle \rangle}{\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle \rangle\!\rangle)}$$



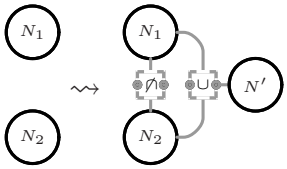
(a) Addition of Inclusion

$$\frac{\begin{array}{l}\langle \langle\!\langle N_1 \rangle\!\rangle, \overset{\mathrm{S}}{\cap}, \langle\!\langle N_2 \rangle\!\rangle \rangle \\ \texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\cap})\end{array}}{\begin{array}{l}\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid \{x\} \leftarrow \langle\!\langle N_1 \rangle\!\rangle; \{x\} \leftarrow \langle\!\langle N_2 \rangle\!\rangle]) \\ \mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle \rangle\!\rangle) \\ \mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle \rangle\!\rangle)\end{array}}$$



(b) Addition of Intersection

$$\frac{\begin{array}{l}\langle \langle\!\langle N_1 \rangle\!\rangle, \overset{\mathrm{S}}{\not\cap}, \langle\!\langle N_2 \rangle\!\rangle \rangle \\ \texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\not\cap})\end{array}}{\begin{array}{l}\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle \texttt{ ++ } \langle\!\langle N_2 \rangle\!\rangle) \\ \mathsf{addConstraint}(\langle\!\langle \cup, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle \rangle\!\rangle) \\ \mathsf{addConstraint}(\langle\!\langle \not\cap, \langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle \rangle\!\rangle)\end{array}}$$



(c) Addition of Union

Figure 6.4: Unioning rules

The first condition of the rule is satisfied by the mapping $\langle\langle\langle\mathsf{paper}\rangle\rangle, \overset{\mathrm{S}}{\not\cap}, \langle\langle\mathsf{book}\rangle\rangle\rangle$. The second condition calls the function `uniqueName`, which in this case could be implemented by concatenating the two names, *e.g.* `uniqueName(paper,book,paperORbook,`$\overset{\mathrm{S}}{\not\cap}$`)`.

The rule then generates the following transformations on schemas S_1'' and S_2''.

$\mathsf{addNode}(\langle\langle\mathsf{paperORbook}\rangle\rangle, \langle\langle\mathsf{paper}\rangle\rangle \ \mathsf{++} \ \langle\langle\mathsf{book}\rangle\rangle)$

$\mathsf{addConstraint}(\langle\langle\cup, \mathsf{paperORbook}, \mathsf{paper}, \mathsf{book}\rangle\rangle)$

$\mathsf{addConstraint}(\langle\langle\not\cap, \mathsf{paper}, \mathsf{book}\rangle\rangle)$

Similarly, the Addition of Union rule can be applied on the disjoint nodes $\langle\langle\mathsf{paper : bibtex}\rangle\rangle$ and $\langle\langle\mathsf{book : id}\rangle\rangle$, the Addition of Inclusion rule can be applied on $\langle\langle\mathsf{paper : year}\rangle\rangle$ which subsumes $\langle\langle\mathsf{book : year}\rangle\rangle$ and, finally, the Addition of Intersection rule can be applied for the intersecting nodes: (a) $\langle\langle\mathsf{paper : title}\rangle\rangle$ and $\langle\langle\mathsf{book : title}\rangle\rangle$, (b) $\langle\langle\mathsf{paper\_author}\rangle\rangle$ and $\langle\langle\mathsf{book\_author}\rangle\rangle$ and (c) $\langle\langle\mathsf{paper\_author : name}\rangle\rangle$ and $\langle\langle\mathsf{book\_author : name}\rangle\rangle$.

The final schema S_{12} produced in the unioning phase is illustrated in Figure 6.5.

◇

6.1.4 Restructuring

In the final restructuring phase [8] of schema merging, the existence of equivalence, subsumption, intersection and disjointness relationships between edges are examined. The set of formally defined rules of this phase includes **Redundant Edge Removal** rules, **Optional Edge Removal**, **Specialization of Edges**, **Addition of Edge Intersection** and **Generalization of Edges** rules. The purpose of these rules is to minimize duplication and simplify the schema. As an illustrative case, in

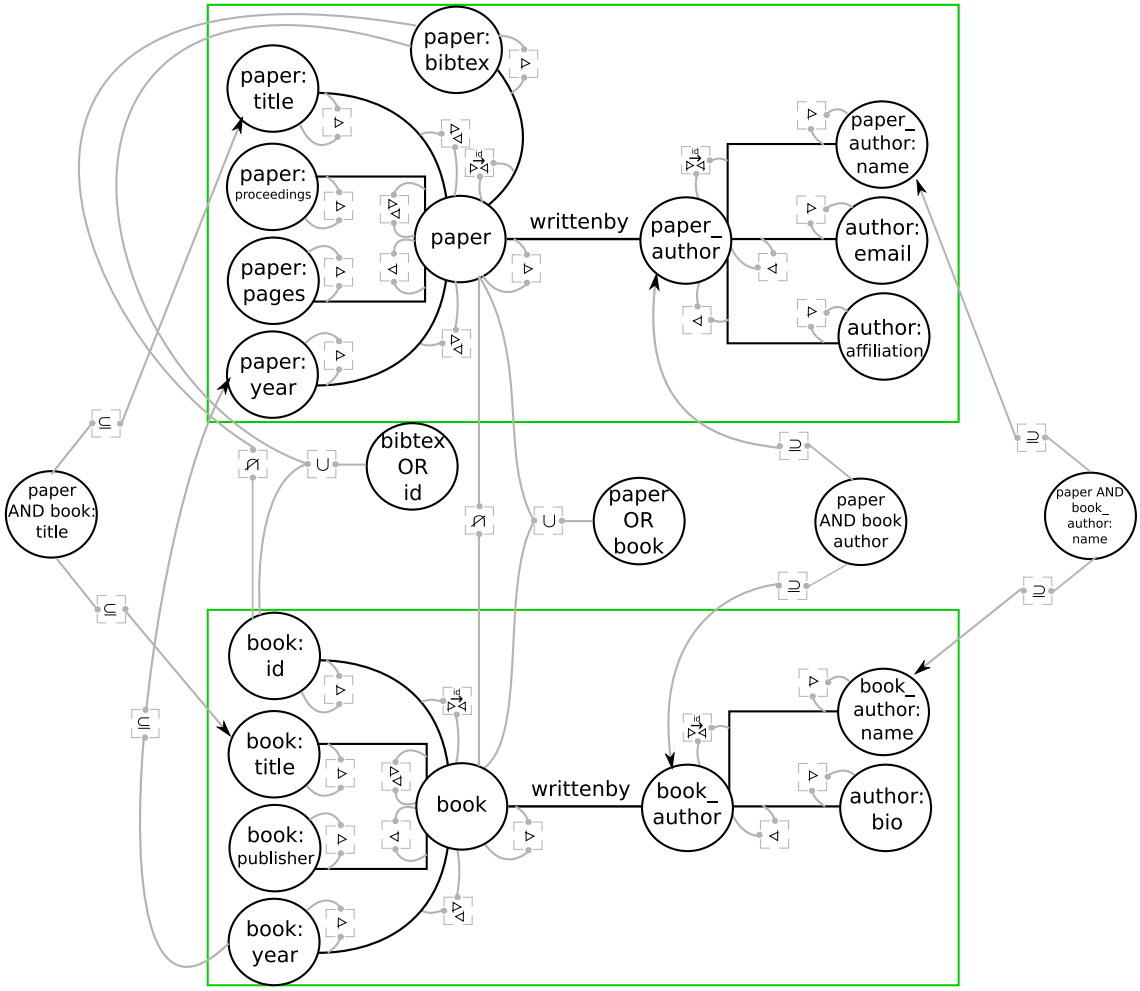


Figure 6.5: Schema S_{12} produced after unioning schemas S_1'' and S_2''. The green boxes just remind the reader how the schemas looked after the naming conforming phase. All objects outside the green boxes have been added during the unioning phase.

this section we are going to consider only rules related to the disjointness relationship between edges. All rules are available in Appendix A.

When a disjointness relationship is identified between two edges e_1 and e_2, then a union edge e' can be added. The new edge will subsume the disjoint edges. The purpose of the rule is to add the union edge if the disjoint edges e_1 and e_2 can then be deleted. Otherwise, the addition of the union edge just produces redundancy. Essentially, what we would like to accomplish is to generalize the disjoint edges into the union edge and then remove the disjoint edges. Thus, the rule that arises in an edge disjointness case, is the **Generalization of Edges** rule.

Before adding the union edge e' the nodes that e' associates have to be added. In the general case, these nodes are the union nodes of the nodes at both ends of e_1 and e_2. In some cases the union nodes are not necessary, because they already exist. For example, if a disjointness relationship has been identified between the corresponding nodes, then during the unioning phase the union nodes would have been added by the Addition of Union rule.

Since, we want to delete the edges e_1 and e_2, we need to pay attention to whether the extent of these edges can be reproduced from the extent of the added union edge e'. If the extents of e_1 and e_2 cannot be *exactly* reproduced then the merging rule causes either information loss or gain. For example, if the disjoint edges e_1 and e_2 are $\langle\langle \mathsf{first\_supervisor}, \mathsf{phd}, \mathsf{academic} \rangle\rangle$ and $\langle\langle \mathsf{second\_supervisor}, \mathsf{phd}, \mathsf{academic} \rangle\rangle$, which associate each PhD student with either her first or second supervisors, and e' is the union edge $\langle\langle \mathsf{supervisor}, \mathsf{phd}, \mathsf{academic} \rangle\rangle$, then we cannot determine based only on e' which instances of the union edge are instances of first_supervisor and second_supervisor. We know that $Ext_{S,I}(\mathsf{first\_supervisor}) \subseteq Ext_{S,I}(\mathsf{supervisor})$ and $Ext_{S,I}(\mathsf{second\_supervisor}) \subseteq Ext_{S,I}(\mathsf{supervisor})$, but we cannot be exact on the extents of the two edges. Therefore in this example, we cannot delete the edges first_supervisor and second_supervisor.

In order to be able to reproduce the extents of $\langle\langle e_1, N_1, N_1'\rangle\rangle$ and $\langle\langle e_2, N_2, N_2'\rangle\rangle$ from the union edge e', we must enforce some constraint. The necessary constraint is that the nodes at one end of the edges must also be disjoint, *i.e.* either $\langle\langle\langle N_1\rangle\rangle, \overset{\text{S}}{\not\cap}, \langle\langle N_2\rangle\rangle\rangle$ or $\langle\langle\langle N_1'\rangle\rangle, \overset{\text{S}}{\not\cap}, \langle\langle N_2'\rangle\rangle\rangle$.

For example, if $\langle\langle\langle N_1\rangle\rangle, \overset{\text{S}}{\not\cap}, \langle\langle N_2\rangle\rangle\rangle$ and $\langle\langle N'\rangle\rangle$ is the union node of $\langle\langle N_1\rangle\rangle$ and $\langle\langle N_2\rangle\rangle$ added during the unioning phase, then the extent of e_1 can be determined from e' by identifying all the instances $\{x, y\}$ of e' where $x \in \text{Ext}_{S,I}(N_1)$, *i.e.* using query $Q_1 = [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N', N''\rangle\rangle; \{x\} \leftarrow \langle\langle N_1\rangle\rangle]$. Similarly, the extent of e_2 can be derived from e' based on the query $Q_2 = [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N', N''\rangle\rangle; \{x\} \leftarrow \langle\langle N_2\rangle\rangle]$. If N_1 and N_2 are not disjoint but they are sharing some instances, then it is possible that query Q_1 contains instances of e_2 and Q_2 contains instances of e_1, *i.e.* $Ext_{S,I}(e_1) \subset Q_1$ and $Ext_{S,I}(e_2) \subset Q_2$. Thus, if N_1 and N_2 are not disjoint, then the queries Q_1 and Q_2 would not retrieve the exact extent of the edges, but they would include additional instances causing an information gain.

In Figure 6.6 a Generalization of Edges rule is defined for mapping $\langle\langle\langle e_1, N_1, N_1'\rangle\rangle$, $\overset{\text{S}}{\not\cap}$, $\langle\langle e_2, N_2, N_2'\rangle\rangle\rangle$. The rule enforces the constraint $\langle\langle\langle N_1\rangle\rangle, \overset{\text{S}}{\not\cap}, \langle\langle N_2\rangle\rangle\rangle$. The rule uses some auxiliary functions:

- `createdNodal(`N_1`,`N_2`,`N'`)`: identifies the node N' created during the unioning phase, due to the relationship between N_1 and N_2. As seen from the unioning rules in Figure 6.4, there is only one such node for each pair N_1 and N_2.

- `constraints(`N`,`e`,`$Constraints$`)`: identifies the constraints $Constraints$ between N and e. Variables N and e have to be grounded.

- `commonCons(`C_1`,`C_2`,`C`)`: identifies the common constraints C between C_1 and C_2. C_1 and C_2 have to be grounded.

- `addConsList(`C`,`N'`,`e'`)`: generates add transformations for the constraints C

between N' and e'

- `genDeleteCons(`*Constraints*`)`: generates delete transformations for the list of constraints *Constraints*.

- `moveDependents(`e_1`,`e_2`)`: replaces each reference to e_1 with a reference to e_2. Thus, any object which depends on e_1 will be now *moved* to depend on e_2.

The rule based on the disjointness relationship of the nodes on one end of the edges identifies the union node N' created during the unioning phase, `createdNodal(`$\langle\langle N_1 \rangle\rangle$, $\langle\langle N_2 \rangle\rangle$, $\langle\langle N' \rangle\rangle$`)`. Thus, the union node for the union edge e' that the rule was going to add already exists from one end of the edges. In the other end, the nodes are intersecting $\langle \langle\langle N'_1 \rangle\rangle, \overset{S}{\cap}, \langle\langle N'_2 \rangle\rangle \rangle$, which results in an intersection node N'_{12} during unioning, `createdNodal(`$\langle\langle N'_1 \rangle\rangle$`,`$\langle\langle N'_2 \rangle\rangle$`,`$\langle\langle N'_{12} \rangle\rangle$`)`. Thus, the rule[1] needs to create the union node for this end of the edge e'. The rule derives the name of this union node to be N'', `uniqueName(`N'_1`,`N'_2`,`N''`,`$\overset{S}{\cap}$`)`, and adds it with its extent to be the union of the extents of N'_1 and N'_2, `addNode(`$\langle\langle N'_1 \rangle\rangle$ `++` $\langle\langle N'_2 \rangle\rangle$`)`. The union edge e' is also added and it associates the two union nodes N' and N'', $\langle\langle e', N', N'' \rangle\rangle$. Any constraints that are common between e_1 and e_2 are added to e' and any other constraints are deleted. Finally, the edges e_1 and e_2 are deleted with associated queries Q_1 and Q_2 respectively, as explained previously.

Example 6.3. Restructuring: Edge Disjointness In our running example, in the restructuring phase the Generalization of Edges rule can be applied on schema S_{12} (Figure 6.5) due to the mapping:
$\langle \langle\langle \mathsf{writtenby}, \mathsf{paper}, \mathsf{paper\_author} \rangle\rangle, \overset{S}{\cap}, \langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{book\_author} \rangle\rangle \rangle$.

The second condition of the rule is satisfied by the mapping $\langle \langle\langle \mathsf{paper} \rangle\rangle, \overset{S}{\cap}, \langle\langle \mathsf{book} \rangle\rangle \rangle$ and the third condition binds $\langle\langle N' \rangle\rangle$ to $\langle\langle \mathsf{paperORbook} \rangle\rangle$. The

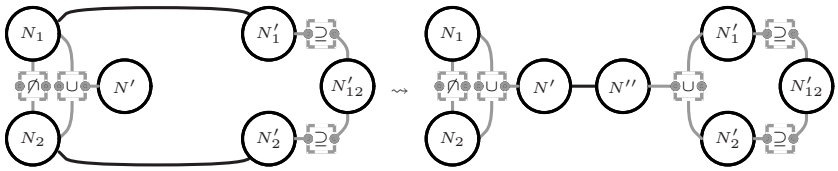
[1]This rule maps to the third Generalization of Edges rule in Appendix A in which the nodes at each end of the disjoint edges have the same semantic relationships (disjointness, intersection) in reverse order.

$\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle \overset{\mathrm{S}}{\sqcap} \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle$
$\langle\!\langle N_1 \rangle\!\rangle \overset{\mathrm{S}}{\sqcap} \langle\!\langle N_2 \rangle\!\rangle$
createdNodal($\langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle N' \rangle\!\rangle$)
$\langle\!\langle N_1' \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle N_2' \rangle\!\rangle$
createdNodal($\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle N_{12}' \rangle\!\rangle$)
uniqueName($N_1', N_2', N'', \overset{\mathrm{S}}{\sqcap}$)
uniqueName($e_1, e_2, e', \overset{\mathrm{S}}{\sqcap}$)
constraints($\langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, Constraints_1$)
constraints($\langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle, Constraints_2$)
commonCons($Constraints_1, Constraints_2, Constraints$)
constraints($\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, Constraints_1'$)
constraints($\langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle, Constraints_2'$)
commonCons($Constraints_1', Constraints_2', Constraints'$)

---

addNode($\langle\!\langle N'' \rangle\!\rangle, \langle\!\langle N_1' \rangle\!\rangle$ ++ $\langle\!\langle N_2' \rangle\!\rangle$)
addEdge($\langle\!\langle e', N', N'' \rangle\!\rangle, \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle]$ ++ $\langle\!\langle e_2, N_2, N_2' \rangle\!\rangle$)
addConsList($Constraints, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle e', N', N'' \rangle\!\rangle$)
addConsList($Constraints', \langle\!\langle N'' \rangle\!\rangle, \langle\!\langle e', N', N'' \rangle\!\rangle$)
genDeleteCons($Constraints_1$)
moveDependents(e_1, e')
deleteEdge($\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\!\langle e', N', N'' \rangle\!\rangle; \{x\} \leftarrow \langle\!\langle N_1 \rangle\!\rangle]$)
genDeleteCons($Constraints_2$)
moveDependents(e_2, e')
deleteEdge($\langle\!\langle e_2, N_2, N_2' \rangle\!\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\!\langle e', N', N'' \rangle\!\rangle; \{x\} \leftarrow \langle\!\langle N_2 \rangle\!\rangle]$)

(a) Formal Rule



(b) Graphical representation

Figure 6.6: Generalization of Edges rule

fourth condition is satisfied by the mapping $\langle\langle\langle\mathsf{paper\_author}\rangle\rangle, \overset{S}{\cap}, \langle\langle\mathsf{book\_author}\rangle\rangle\rangle$ and the fifth condition binds $\langle\langle N'_{12}\rangle\rangle$ to $\langle\langle\mathsf{paperANDbookauthor}\rangle\rangle$.

Then a new name for the union node for the right hand side of the edges is identified, binding N'' to author, and a new name for the union edge is identified, binding e' to writtenby.

Variables $Constraints_1$ and $Constraints_2$ get value $[\triangleright]$ for the mandatory constraint between $\langle\langle\mathsf{paper}\rangle\rangle$ and $\langle\langle\mathsf{writtenby, paper, paper\_author}\rangle\rangle$, and the mandatory constraint between $\langle\langle\mathsf{book}\rangle\rangle$ and $\langle\langle\mathsf{writtenby, book, book\_author}\rangle\rangle$, respectively. Variable $Constraints$ which identifies the common constraints between $Constraints_1$ and $Constraints_2$ gets values $[\triangleright]$, as well.

On the right hand side, variables $Constraints'_1$ and $Constraints'_2$ get the value of an empty list [ ] and thus $Constraints'$ also gets value [ ].

The rule then generates the following transformations on S_{12}:

addNode($\langle\langle$author$\rangle\rangle$, $\langle\langle$paper_author$\rangle\rangle$++$\langle\langle$book_author$\rangle\rangle$)

addEdge($\langle\langle$writtenby, paperORbook, author$\rangle\rangle$,

$\langle\langle$writtenby, paper, paper_author$\rangle\rangle$ ++ $\langle\langle$writtenby, book, book_author$\rangle\rangle$)

addConstraint($\langle\langle\triangleright, \langle\langle$paperORbook$\rangle\rangle, \langle\langle$writtenby, paperORbook, author$\rangle\rangle\rangle\rangle$)

deleteConstraint($\langle\langle\triangleright, \langle\langle$paper$\rangle\rangle, \langle\langle$writtenby, paper, paper_author$\rangle\rangle\rangle\rangle$)

deleteEdge($\langle\langle$writtenby, paper, paper_author$\rangle\rangle$,

$\{x, y\} \mid \{x, y\} \leftarrow \langle\langle$writtenby, paperORbook, author$\rangle\rangle; \{x\} \leftarrow \langle\langle$paper$\rangle\rangle$)

deleteConstraint($\langle\langle\triangleright, \langle\langle$book$\rangle\rangle, \langle\langle$writtenby, book, book_author$\rangle\rangle\rangle\rangle$)

deleteEdge($\langle\langle$writtenby, book, book_author$\rangle\rangle$,

$\{x, y\} \mid \{x, y\} \leftarrow \langle\langle$writtenby, paperORbook, author$\rangle\rangle; \{x\} \leftarrow \langle\langle$book$\rangle\rangle$)

The final integrated schema is illustrated in Figure 6.7.

$\diamond$

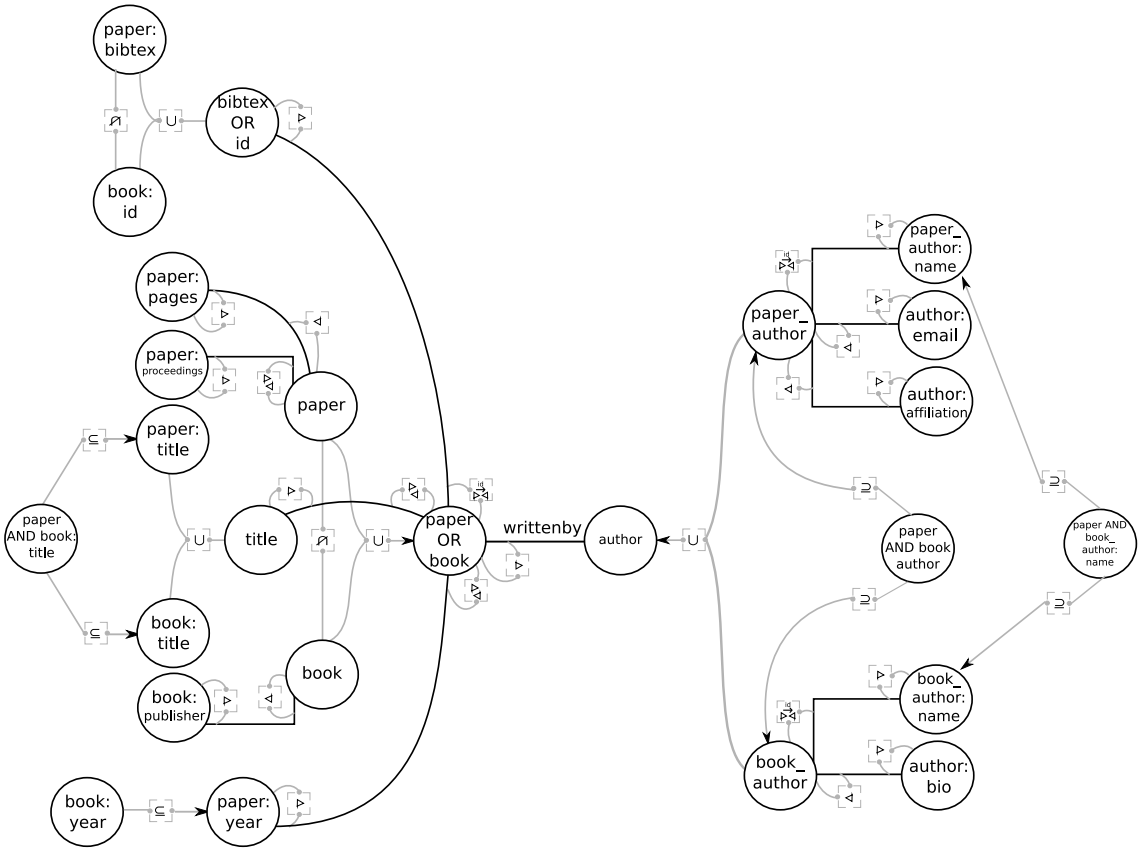


Figure 6.7: The final result of the integration of schemas S_1 and S_2

6.1.5 Properties of Low-Level Merging

In this section we are going to discuss the properties of our proposed merging rules and the properties of the resulting schemas.

During the naming conforming phase, the Node and Edge Merge rules collapse each pair of equivalent objects into a single object. The Node and Edge Distinction rules explicitly differentiate each two non-equivalent objects. Therefore, after the naming conforming and the superimposition of the schemas, in the resulting schema:

1. objects are preserved. For each object in the initial schemas an object exists in the resulting schema.
2. non-equivalence is preserved. For each pair of objects which are non-equivalent in the initial schemas, they are distinct in the resulting schema.
3. duplicates are eliminated. The resulting schema is duplicate-free, *i.e.* there are no two distinct objects which are semantically equivalent.

For the last property to hold, the initial schemas must also be duplicate-free.

During the unioning phase, the Addition of Intersection and Union rules add nodes onto the schema, therefore duplicate nodes might be created. For example, assume one schema that contains nodes $\langle\langle \mathsf{ug} \rangle\rangle$ and $\langle\langle \mathsf{student} \rangle\rangle$ and another schema that contains node $\langle\langle \mathsf{pg} \rangle\rangle$. A disjointness relationship is identified between nodes $\langle\langle \mathsf{ug} \rangle\rangle$ and $\langle\langle \mathsf{pg} \rangle\rangle$, which represent undergraduate and postgraduate students. Therefore, when the Addition of Union rule creates the union node $\langle\langle N' \rangle\rangle$ of $\langle\langle \mathsf{ug} \rangle\rangle$ and $\langle\langle \mathsf{pg} \rangle\rangle$, this node will be equivalent to $\langle\langle \mathsf{student} \rangle\rangle$. Thus, two equivalent but distinct nodes, $\langle\langle N' \rangle\rangle$ and $\langle\langle \mathsf{student} \rangle\rangle$, will be part of the resulting schema. The same problem arises in the restructuring phase of schema merging, where there are rules that add nodes and/or edges.

This problem can be solved either manually or automatically. In the first case, the user could intervene and specify the transformations that should not be applied

because they create duplicate objects. In the latter case, schema matching could be performed on the final integrated schema. Matching could identify the equivalent objects which can then be collapsed into single objects by applying the Node and Edge Merge rules. Because of the constraints and the previously identified semantic mappings, during this schema matching process there are few objects that need to be compared. For example, only the added union node $\langle\langle N' \rangle\rangle$ of $\langle\langle \mathsf{ug} \rangle\rangle$ and $\langle\langle \mathsf{pg} \rangle\rangle$ needs to matched against existing objects, and additionally it does not need to be compared with all the objects but just the ones which subsume either $\langle\langle \mathsf{ug} \rangle\rangle$ and/or $\langle\langle \mathsf{pg} \rangle\rangle$.

The most important property of our merging rules is what we call the **Intentional Domain Preservation Property** (**IDPP**), which is also referred to as *completeness* and *minimality* in the literature [87]. If S_1 and S_2 are the schemas before and after the application of a merging rule r, then r conforms to the IDPP if for each instance of an object of S_1, this instance can be derived from the instances of the objects in S_2, and *vice versa*. Thus, no information is lost or gained when transforming S_1 to S_2 and *vice versa*. In this case, we say that the rule r is both **complete** and **sound**. Any rules whose actions contain only add, delete and rename transformations can be considered as possibly obeying the IDPP, provided their IQL queries have been correctly formulated. This is by definition of the add, delete and rename transformations, which specify the exact extent of the object added, deleted and renamed, respectively, *i.e.* they do not allow any information loss or gain apart from what is identified by their associated IQL query. The completeness and soundness properties of our rules demonstrate that the queries have been correctly formulated.

For example, consider the Generalization of Edges rule in Figure 6.6. In order to check that the rule is both complete and sound we need to examine its transformations. The interesting cases are the deletion of edges e_1 and e_2.

We can show that edge e_1 can be deleted because its extent can be exactly reproduced from e', as specified by the query attached to the $\mathsf{delete}\ e_1$ transformation. Lemmas 2

and 3 state this formally. The same holds for the deletion of edge e_2. The proof of Lemma 2 is supplied below as an illustrative example of how rules are examined for their completeness and soundness.

Lemma 2. Let e_1, e_2 be two edges.

$$
\begin{array}{ll}
\text{if} & \langle\langle\langle e_1, N_1, X_1\rangle\rangle, \overset{\text{S}}{\not\cap}, \langle\langle e_2, N_2, X_2\rangle\rangle\rangle, \\
& \langle\langle\langle N_1\rangle\rangle, \overset{\text{S}}{\not\cap}, \langle\langle N_2\rangle\rangle\rangle, \\
& \texttt{createdNodal(}\langle\langle N_1\rangle\rangle\texttt{,}\langle\langle N_2\rangle\rangle\texttt{,}\langle\langle N'\rangle\rangle\texttt{)}, \\
& Ext_{S,I}(\langle\langle e', N', X\rangle\rangle) = \\
& \qquad Ext_{S,I}(\langle\langle e_1, N_1, X_1\rangle\rangle) \cup Ext_{S,I}(\langle\langle e_2, N_2, X_2\rangle\rangle). \\
& Ext_{S,I}(\langle\langle X\rangle\rangle) = Ext_{S,I}(\langle\langle X_1\rangle\rangle) \cup Ext_{S,I}(\langle\langle X_2\rangle\rangle) \\
\text{then} & \forall x, y.\, \{x, y\} \in Ext_{S,I}(\langle\langle e', N', X\rangle\rangle), \\
& \qquad \{x\} \in Ext_{S,I}(\langle\langle N_1\rangle\rangle) \\
& \quad \rightarrow \{x, y\} \in Ext_{S,I}(\langle\langle e_1, N_1, N_1'\rangle\rangle)
\end{array}
$$

In the Generalization of Edges rule (Figure 6.6) all the conditions in the lemma hold. The lemma shows that each instance retrieved by the query in the delete e_1 transformation $[\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N', N''\rangle\rangle; \{x\} \leftarrow \langle\langle N_1\rangle\rangle]$ is an instance of e_1. Thus the transformation is sound. Before we give the proof of the lemma above we need to define the **cartesian extent** of an edge.

Definition 6.1. Cartesian Extent The cartesian extent of an edge $\langle\langle e, N_1, N_2\rangle\rangle$ is the cartesian product of the extents of the associated nodes in e, *i.e.*
$Ext_{S,I,\times}(\langle\langle e, N_1, N_2\rangle\rangle) = Ext_{S,I}(\langle\langle N_1\rangle\rangle) \times Ext_{S,I}(\langle\langle N_2\rangle\rangle)$.

□

Now, we can prove Lemma 2.

Proof. By the definition of disjointness $\langle\langle\langle N_1\rangle\rangle, \overset{\text{S}}{\not\cap}, \langle\langle N_2\rangle\rangle\rangle$ we have:

$$
\begin{array}{ll}
\forall x_1, x_2. & \{x_1\} \in Ext_{S,I}(\langle\langle N_1\rangle\rangle), \{x_2\} \in Ext_{S,I}(\langle\langle N_2\rangle\rangle) \\
\quad \rightarrow & \{x_1\} \neq \{x_2\}.
\end{array}
$$

We may add redundant terms to the above implication as follows:

$$\begin{aligned}\forall x_1, x_2, \forall y_1, y_2. \quad & \{x_1\} \in Ext_{S,I}(\langle\langle N_1 \rangle\rangle), \{x_2\} \in Ext_{S,I}(\langle\langle N_2 \rangle\rangle), \\ & \{y_1\} \in Ext_{S,I}(\langle\langle X \rangle\rangle), \{y_2\} \in Ext_{S,I}(\langle\langle X_2 \rangle\rangle) \\ \rightarrow \quad & \{x_1, y_1\} \neq \{x_2, y_2\}.\end{aligned}$$

which based on the definition of the cartesian extent can be written:

$$\begin{aligned}\forall x_1, x_2, \forall y_1, y_2. \quad & \{x_1, y_1\} \in Ext_{S,I,\times}(\langle\langle \_, N_1, X \rangle\rangle), \{x_2, y_2\} \in Ext_{S,I,\times}(\langle\langle e_2, N_2, X_2 \rangle\rangle) \\ \rightarrow \quad & \{x_1, y_1\} \neq \{x_2, y_2\}.\end{aligned}$$

i.e. $Ext_{S,I,\times}(\langle\langle \_, N_1, X \rangle\rangle) \cap Ext_{S,I,\times}(\langle\langle e_2, N_2, X_2 \rangle\rangle) = \emptyset$.

By definition of the cartesian extent of an edge, we have $Ext_{S,I}(\langle\langle e_2, N_2, X_2 \rangle\rangle) \subseteq Ext_{S,I,\times}(\langle\langle e_2, N_2, X_2 \rangle\rangle)$, and thus

$$Ext_{S,I,\times}(\langle\langle \_, N_1, X \rangle\rangle) \cap Ext_{S,I}(\langle\langle e_2, N_2, X_2 \rangle\rangle) = \emptyset \tag{6.1}$$

Since $Ext_{S,I}(X) = Ext_{S,I}(X_1) \cup Ext_{S,I}(X_2)$, we have that $Ext_{S,I}(X_1) \subseteq Ext_{S,I}(X)$, and therefore $Ext_{S,I}(\langle\langle e_1, N_1, X_1 \rangle\rangle) \subseteq Ext_{S,I,\times}(\langle\langle \_, N_1, X \rangle\rangle)$. Thus,

$$Ext_{S,I}(\langle\langle e_1, N_1, X_1 \rangle\rangle) \cap Ext_{S,I,\times}(\langle\langle \_, N_1, X \rangle\rangle) = Ext_{S,I}(\langle\langle e_1, N_1, X_1 \rangle\rangle) \tag{6.2}$$

Based on the definition of the extent of an edge, we have

$$\begin{aligned}\forall x, y. \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e', N', X \rangle\rangle), \{x\} \in Ext_{S,I}(\langle\langle N_1 \rangle\rangle) \\ \rightarrow \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e', N', X \rangle\rangle), \{x\} \in Ext_{S,I}(\langle\langle N' \rangle\rangle), \\ & \{y\} \in Ext_{S,I}(\langle\langle X \rangle\rangle), \{x\} \in Ext_{S,I}(\langle\langle N_1 \rangle\rangle) \\ \rightarrow \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e', N', X \rangle\rangle), \\ & \{x\} \in Ext_{S,I}(\langle\langle N' \rangle\rangle) \cap Ext_{S,I}(\langle\langle N_1 \rangle\rangle), \\ & \{y\} \in Ext_{S,I}(\langle\langle X \rangle\rangle)\end{aligned}$$

Based on the initial conditions we have that $Ext_{S,I}(\langle\langle N_1 \rangle\rangle) \subseteq Ext_{S,I}(\langle\langle N' \rangle\rangle)$ and that $Ext_{S,I}(\langle\langle e', N', X \rangle\rangle) = Ext_{S,I}(\langle\langle e_1, N_1, X_1 \rangle\rangle) \cup Ext_{S,I}(\langle\langle e_2, N_2, X_2 \rangle\rangle)$, thus the above implication can be re-written

$$\begin{aligned}
\forall x, y. \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e', N', X\rangle\rangle), \{x\} \in Ext_{S,I}(\langle\langle N_1\rangle\rangle) \\
\rightarrow \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e_1, N_1, X_1\rangle\rangle) \cup Ext_{S,I}(\langle\langle e_2, N_2, X_2\rangle\rangle) \\
& \{x\} \in Ext_{S,I}(\langle\langle N_1\rangle\rangle), \{y\} \in Ext_{S,I}(\langle\langle X\rangle\rangle)
\end{aligned}$$

which based on the cartesian extent definition becomes

$$\begin{aligned}
\forall x, y. \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e', N', X\rangle\rangle), \{x\} \in Ext_{S,I}(\langle\langle N_1\rangle\rangle) \\
\rightarrow \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e_1, N_1, X_1\rangle\rangle) \cup Ext_{S,I}(\langle\langle e_2, N_2, X_2\rangle\rangle) \\
& \{x, y\} \in Ext_{S,I,\times}(\langle\langle _, N_1, X\rangle\rangle) \\
\rightarrow \quad & \{x, y\} \in (Ext_{S,I}(\langle\langle e_1, N_1, X_1\rangle\rangle) \cup Ext_{S,I}(\langle\langle e_2, N_2, X_2\rangle\rangle)) \cap \\
& Ext_{S,I,\times}(\langle\langle _, N_1, X\rangle\rangle) \\
\rightarrow \quad & \{x, y\} \in (Ext_{S,I}(\langle\langle e_1, N_1, X_1\rangle\rangle) \cap Ext_{S,I,\times}(\langle\langle _, N_1, X\rangle\rangle)) \cup \\
& (Ext_{S,I}(\langle\langle e_2, N_2, X_2\rangle\rangle) \cap Ext_{S,I,\times}(\langle\langle _, N_1, X\rangle\rangle))
\end{aligned}$$

Finally, the last implication based on Equations 6.1 and 6.2 becomes

$$\begin{aligned}
\forall x, y. \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e', N', X\rangle\rangle), \{x\} \in Ext_{S,I}(\langle\langle N_1\rangle\rangle) \\
\rightarrow \quad & \{x, y\} \in Ext_{S,I}(\langle\langle e_1, N_1, X_1\rangle\rangle)
\end{aligned}$$

□

Regarding the completeness of the deletion of e_1, we need to show that each instance of e_1 is an instance retrieved by the query in the delete e_1 transformation. This is the inverse of what the previous lemma stated, and it is formally explained below.

Lemma 3. Let e_1, e_2 be two edges.

$$\begin{aligned}
\text{if} \quad & \langle\langle e_1, N_1, X_1\rangle\rangle, \overset{\text{S}}{\emptyset}, \langle\langle e_2, N_2, X_2\rangle\rangle\rangle, \\
& Ext_{S,I}(\langle\langle e', N', X\rangle\rangle) = Ext_{S,I}(\langle\langle e_1, N_1, X_1\rangle\rangle) \cup Ext_{S,I}(\langle\langle e_2, N_2, X_2\rangle\rangle). \\
\text{then} \quad & \forall x, y. \{x, y\} \in Ext_{S,I}(\langle\langle e_1, N_1, X_1\rangle\rangle) \\
& \quad \rightarrow \{x, y\} \in Ext_{S,I}(\langle\langle e', N', X\rangle\rangle), \\
& \quad\quad \{x\} \in Ext_{S,I}(\langle\langle N_1\rangle\rangle)
\end{aligned}$$

which can be trivially proved based on the facts that

$Ext_{S,I}(e_1, N_1, X_1) \subseteq Ext_{S,I}(\langle\langle e', N', X\rangle\rangle)$, and
$\forall\{x, y\} \in Ext_{S,I}(e_1, N_1, X_1) : \{x\} \in Ext_{S,I}(\langle\langle N_1\rangle\rangle)$

6.1.6 Automatic Schema Merging

In Table 6.1, we have exhaustively identified the possible merging rules. In this section, we identify the degree to which our merging rules can be automated by examining the auxiliary functions that are used in the rules.

First of all, the generated BAV transformations do not need the user's assistance to be performed; the objects and queries associated with them are formally specified. This also holds for the high-level actions, *i.e.* the functions `constraints/3`, `commonCons/3`, `genDeleteCons/3` and `moveDependents/2` of the restructuring phase.

The interesting functions are the name functions used when a new object is added onto the schema. These are: `commonName/3`, `distinctNames/4` and `uniqueName/4`. A possible fully automated implementation of the `commonName/3` function, can identify the common substring of the names of two objects. Another implementation could use a preferred schema, where the object names of the preferred schema are used, *e.g.* in a data exchange scenario the preferred schema would be the target schema.

Function `distinctNames/4` can also be automated quite simply. It takes as input two identical names of two objects in different schemas and returns two distinct names. The distinct names can be produced by simply prefixing the name of the objects with the name of the schema they belong to, or with the name of the schema object they are attached to (Example 6.1).

Similarly, the `uniqueName/4` function can be fully automated by identifying a random name or by concatenating the names of the objects the rule is dealing with. In some cases, such implementations might be sufficient, *e.g.* in a meta-search engine

where the merged schema is not presented to the user. On the other hand, if the schema is available to the user or it is used in some other schema integration scenario, such a fully automated implementation might be problematic, since the names produced are ambiguous hampering the understanding of the integrated schema.

In general, our proposed schema merging approach is semi-automatic with the minimum user's effort necessary: the integrated schema can be produced automatically but the user might have to perform a few rename transformations to assign more intelligent names to the schema objects.

6.2 Generic Schema Merging

Based on our low-level schema merging introduced in Section 6.1 and using the generic classification of high-level model constructs introduced in Section 3.3, we can define a generic schema merging methodology, independent of any data model used. We remind the reader that the generic classification of constructs defines four classification types: nodal, link, link-nodal and constraint constructs. Additionally, we know that the nodal construct maps to the HDM node, the link construct maps to the HDM edge and that the link-nodal construct maps to an edge, node and constraints combination in the HDM. Note that in this and the following sections, the notation of schema objects is prefixed with the data model of the object for clarity.

Similarly to our low-level merging, generic merging uses a set of formally defined rules and follows the three integration phases of naming conforming, unioning and restructuring. The rules of our generic merging are derived from their respective low-level rules but they are driven based on the semantic relationships between nodal, link, and link-nodal objects. The rules can then be translated into high-level model specific rules as will be explained in Section 6.3.

During naming conforming, generic merging specifies **Nodal Merge** and **Distinction** rules, **Link Merge** and **Distinctions** rules, and **Link-Nodal Merge** and **Distinction** rules. Since the nodal construct maps to the HDM node, the rules about nodes in low-level merging can be directly mapped to generic nodal rules. For example, the Nodal Merge rule is produced by using the Node Merge rule (Figure 6.1(a)) and just replacing the node BAV transformations to generic nodal BAV transformations:

$$\frac{\begin{array}{l}\langle\langle N_1\rangle\rangle, \stackrel{S}{=}, \langle\langle N_2\rangle\rangle\rangle \\ \neg\ \texttt{identicalNames}(N_1, N_2) \\ \texttt{commonName}(N_1, N_2, N')\end{array}}{\begin{array}{l}\texttt{renameNodal}_{gen}(\langle\langle N_1\rangle\rangle, \langle\langle N'\rangle\rangle) \\ \texttt{renameNodal}_{gen}(\langle\langle N_2\rangle\rangle, \langle\langle N'\rangle\rangle)\end{array}}$$

This rule and all other generic merging rules use generic BAV transformations. For example, the generic $\texttt{renameNodal}_{gen}$ BAV transformation renames a nodal object, $\texttt{renameLN}_{gen}$ renames a link-nodal object, $\texttt{addLN}_{gen}$ adds a link-nodal and $\texttt{deleteLN}_{gen}$ deletes a link-nodal object.

Similarly, the rules about edges in the low-level merging can be directly mapped to generic link rules. The naming conforming rules for link-nodals are also straightforward. The rules are illustrated in Figure 6.8. The most interesting cases of generic merging are the link-nodal rules of the restructuring phase. All rules are available in Appendix B.

Looking at the link-nodal rules in more detail, we first see that a link-nodal object $\langle\langle X, N\rangle\rangle$ maps to a node $\mathsf{hdm}{:}\langle\langle N\rangle\rangle$, an anonymous edge $\mathsf{hdm}{:}\langle\langle \_, X, N\rangle\rangle$ and a mandatory constraint $\mathsf{hdm}{:}\langle\langle N\rangle\rangle \rhd \mathsf{hdm}{:}\langle\langle \_, X, N\rangle\rangle$. Any semantic relationship rel between two link-nodals $\langle\langle X_1, N_1\rangle\rangle, rel, \langle\langle X_2, N_2\rangle\rangle\rangle$ specifies the same relationship rel between the HDM edges the link-nodals consist of: $\langle \mathsf{hdm}{:}\langle\langle \_, X_1, N_1\rangle\rangle, rel, \mathsf{hdm}{:}\langle\langle \_, X_2, N_2\rangle\rangle\rangle$.

$$\frac{\langle\langle X, N_1\rangle\rangle, \stackrel{S}{=}, \langle\langle X, N_2\rangle\rangle\rangle \quad \neg\ \texttt{identicalNames}(N_1, N_2) \quad \texttt{commonName}(N_1, N_2, N')}{\texttt{renameLN}_{gen}(\langle\langle X, N_1\rangle\rangle, \langle\langle X, N'\rangle\rangle) \quad \texttt{renameLN}_{gen}(\langle\langle X, N_2\rangle\rangle, \langle\langle X, N'\rangle\rangle)}$$

(a) Merge

$$\frac{\neg\ \langle\langle X, N_1\rangle\rangle, \stackrel{S}{=}, \langle\langle X, N_2\rangle\rangle\rangle \quad \texttt{identicalNames}(N_1, N_2) \quad \texttt{distinctNames}(N_1, N_2, N_1', N_2')}{\texttt{renameLN}_{gen}(\langle\langle X, N_1\rangle\rangle, \langle\langle X, N_1'\rangle\rangle) \quad \texttt{renameLN}_{gen}(\langle\langle X, N_2\rangle\rangle, \langle\langle X, N_2'\rangle\rangle)}$$

(b) Distinction

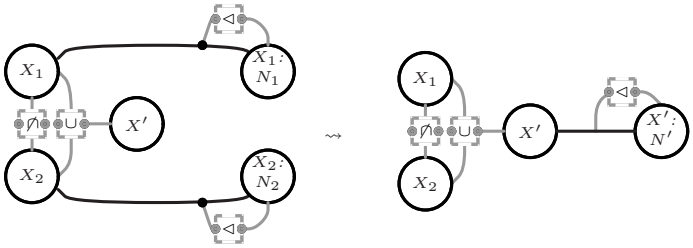
Figure 6.8: Link-Nodal Merge and Distinction rules

This semantic mapping between the edges could potentially trigger the execution of a Redundant Edge Removal, Specialization of Edges, Optional Edge Removal, Addition of Intersection or Generalization of Edges rule during the low-level restructuring phase. The rule r_l that is triggered based on $\langle \mathsf{hdm}{:}\langle\langle \_, X_1, N_1\rangle\rangle, rel, \mathsf{hdm}{:}\langle\langle \_, X_2, N_2\rangle\rangle\rangle$ is the rule that needs to be manipulated so that it is turned into a generic rule r_g for the link-nodals $\langle\langle X_1, N_1\rangle\rangle$ and $\langle\langle X_2, N_2\rangle\rangle$. If r_l deletes one of the edges $\mathsf{hdm}{:}\langle\langle \_, X_1, N_1\rangle\rangle$ and/or $\mathsf{hdm}{:}\langle\langle \_, X_2, N_2\rangle\rangle$, then the r_g deletes together with the edge the node N that the edge is attached to. The reason the node can be deleted is because of the mandatory constraint $\mathsf{hdm}{:}\langle\langle N\rangle\rangle \rhd \mathsf{hdm}{:}\langle\langle \_, X, N\rangle\rangle$, which states that each instance of $\mathsf{hdm}{:}\langle\langle N\rangle\rangle$ appears in the extent of $\mathsf{hdm}{:}\langle\langle \_, X, N\rangle\rangle$. Thus, if the edge $\mathsf{hdm}{:}\langle\langle \_, X, N\rangle\rangle$ can be deleted and its extent can be derived from the remaining objects, as it can be proved for r_l, then the node $\mathsf{hdm}{:}\langle\langle N\rangle\rangle$ can also be deleted since its extent can be derived from the edge.

For example, consider the case of the existence of a disjointness relationship between link-nodal objects $\langle\langle X_1, N_1\rangle\rangle$, $\langle\langle X_2, N_2\rangle\rangle$ when $\langle\langle X_1\rangle\rangle$,$\langle\langle X_2\rangle\rangle$ are also disjoint. Translating this case into the HDM, we have that the edges $\mathsf{hdm}{:}\langle\langle \_, X_1, N_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle \_, X_2, N_2\rangle\rangle$ are disjoint and that the nodes $\mathsf{hdm}{:}\langle\langle X_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle X_2\rangle\rangle$ are disjoint. We cannot deduce anything about the relationship between the nodes $\mathsf{hdm}{:}\langle\langle N_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle N_2\rangle\rangle$. Therefore, to derive the generic rule for this particular case of disjoint link-nodals we could use for example the Generalization of Edges rule in Figure 6.6, where the aforementioned relationships between the HDM ob-

$$\frac{\begin{array}{l}\langle\langle X_1, N_1\rangle\rangle, \overset{\text{S}}{\cap}, \langle\langle X_2, N_2\rangle\rangle\rangle \\ \langle\langle X_1\rangle\rangle, \overset{\text{S}}{\cap}, \langle\langle X_2\rangle\rangle\rangle \\ \texttt{createdNodal}(X_1, X_2, X') \\ \texttt{uniqueName}(N_1, N_2, N', \overset{\text{S}}{\cap})\end{array}}{\begin{array}{l}\texttt{addLN}_{gen}(\langle\langle X', N'\rangle\rangle, [\langle\langle X_1, N_1\rangle\rangle] \texttt{++} [\langle\langle X_2, N_2\rangle\rangle], \\ \texttt{deleteLN}_{gen}(\langle\langle X_1, N_1\rangle\rangle, [\{x, y\} \mid \\ \quad \{x, y\} \leftarrow \langle\langle X', N'\rangle\rangle; \{x\} \leftarrow \langle\langle X_1\rangle\rangle]) \\ \texttt{deleteLN}_{gen}(\langle\langle X_2, N_2\rangle\rangle, [\{x, y\} \mid \\ \quad \{x, y\} \leftarrow \langle\langle X', N'\rangle\rangle; \{x\} \leftarrow \langle\langle X_2\rangle\rangle])\end{array}}$$

(a) Formal Rule



(b) Graphical representation

Figure 6.9: Generalization of Link-Nodals rule

jects hold. In that rule, the union node of $\mathsf{hdm}{:}\langle\langle N_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle N_2\rangle\rangle$ is added, the union edge of $\mathsf{hdm}{:}\langle\langle \_, X_1, N_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle \_, X_2, N_2\rangle\rangle$ is added, and finally the edges $\mathsf{hdm}{:}\langle\langle \_, X_1, N_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle \_, X_2, N_2\rangle\rangle$ are deleted[2]. Based on the low-level rule, the generic rule is also going to: (a) add the union node and the union edge using an $\texttt{addLN}_{gen}$ transformation, and (b) delete the edges $\mathsf{hdm}{:}\langle\langle \_, X_1, N_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle \_, X_2, N_2\rangle\rangle$, and as we explained earlier, also delete the nodes $\mathsf{hdm}{:}\langle\langle N_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle N_2\rangle\rangle$ using $\texttt{deleteLN}_{gen}$ transformations. The result of these transformations is the **Generalization of Link-Nodals** rule illustrated in Figure 6.9.

[2]The node $\mathsf{hdm}{:}\langle\langle N'_{12}\rangle\rangle$ in the Generalization of Edges rule has been added during unioning because of the intersection relationship between $\mathsf{hdm}{:}\langle\langle N_1\rangle\rangle$ and $\mathsf{hdm}{:}\langle\langle N_2\rangle\rangle$. However, in this case of disjoint link-nodals such a node is not created during unioning

6.3 High-level Schema Merging

Based on the generic schema merging methodology described in the previous section, schemas of high-level data models can be integrated. Each generic rule produced by our methodology can be translated into a high-level model specific rule, using techniques from [70, 16]. In this section, we describe our translation methodology and give illustrative examples of generic to ER model rule translation. The schemes of the ER schema objects in this section follow the definitions in Section 3.4.

We have identified four cases of generic rule to specific rule translation:

1. **Exact Translation**: the generic rule can be translated into a model-specific rule by performing a one to one mapping between the generic constructs and transformations and their model-specific equivalents, *e.g.* an $\texttt{renameLN}_{gen}$ transformation in a generic rule would map into an renameAttribute transformation in the corresponding ER model rule.

2. **Model Limitations**: in some cases the translation of a generic rule using a high level data model cannot be exact because a construct or a transformation in the generic rule does not have an equivalent construct or transformation in the high level language. Therefore, some conditions and/or actions of a generic rule might not be translatable. For example, the HDM exclusion constraint cannot be individually modelled in the ER model, and therefore the addition of such a constraint cannot be translated in a ER model rule.

3. **Meta-Constraint Requirements**: because some data models have **meta-constraints**, extra conditions and actions might be necessary for the translation of a generic rule into a model-specific rule. For example, a meta-constraint of the relational model [27] is the existence of a column for every table. Therefore, a column must be added by the relational model rules for every table that they add.

$$\frac{\langle \mathsf{er}{:}\langle\!\langle X, N_1\rangle\!\rangle, \overset{\mathrm{S}}{=}, \mathsf{er}{:}\langle\!\langle X, N_2\rangle\!\rangle\rangle \quad \neg\ \texttt{identicalNames}(N_1, N_2) \quad \texttt{commonName}(N_1, N_2, N')}{\mathsf{renameAttribute}(\mathsf{er}{:}\langle\!\langle X, N_1\rangle\!\rangle, \mathsf{er}{:}\langle\!\langle X, N'\rangle\!\rangle) \quad \mathsf{renameAttribute}(\mathsf{er}{:}\langle\!\langle X, N_2\rangle\!\rangle, \mathsf{er}{:}\langle\!\langle X, N'\rangle\!\rangle)}$$

(a) Merge

$$\frac{\neg\ \langle \mathsf{er}{:}\langle\!\langle X, N_1\rangle\!\rangle, \overset{\mathrm{S}}{=}, \mathsf{er}{:}\langle\!\langle X, N_2\rangle\!\rangle\rangle \quad \texttt{identicalNames}(N_1, N_2) \quad \texttt{distinctNames}(N_1, N_2, N_1', N_2')}{\mathsf{renameAttribute}(\mathsf{er}{:}\langle\!\langle N_1\rangle\!\rangle, \mathsf{er}{:}\langle\!\langle N_1'\rangle\!\rangle) \quad \mathsf{renameAttribute}(\mathsf{er}{:}\langle\!\langle N_2\rangle\!\rangle, \mathsf{er}{:}\langle\!\langle N_2'\rangle\!\rangle)}$$

(b) Distinction

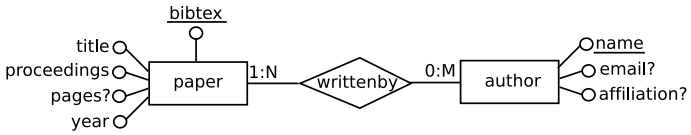
Figure 6.10: Attribute Merge and Distinction rules for the ER model

4. **Meta-Constraint Restrictions**: conditions and/or actions of a generic rule might be restricted in the translated model-specific rule, if they violate the meta-constraints of the data model the rule is translated into, *e.g.* the deletion of a link-nodal construct in a generic rule might be restricted by the corresponding ER model rule, if the link-nodal is a key attribute.

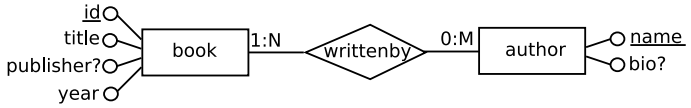
We can now apply our translation methodology for deriving the ER merging rules. The naming conforming rules for the ER model can be produced from the generic rules by Exact Translation. The Nodal Merge and Distinction rules are translated into **Entity Merge** and **Distinction** rules for the ER model. The Link-Nodal Merge and Distinction rules are translated into **Attribute Merge** and **Distinction** rules for the ER model. Figure 6.10 shows as an illustrative example the Attribute Merge and Distinction rules for the ER model, which have been produced by just translating the $\texttt{renameLN}_{gen}$ transformations to renameAttribute transformations.

Example 6.4. ER Naming Conforming Using the example of the introduction, we remind the reader the ER schemas S_1^{er} and S_2^{er} to be integrated in Figure 6.11.

Consider that all compatibility mappings between S_1^{er} and S_2^{er} objects are the ones listed below:



(a) S_1^{er}



(b) S_2^{er}

Figure 6.11: ER schemas S_1^{er} and S_2^{er}

$\langle \mathsf{er}{:}\langle\langle \mathsf{paper} \rangle\rangle, \overset{\mathrm{S}}{\not\cap}, \mathsf{er}{:}\langle\langle \mathsf{book} \rangle\rangle \rangle$

$\langle \mathsf{er}{:}\langle\langle \mathsf{paper}, \mathsf{bibtex} \rangle\rangle, \overset{\mathrm{S}}{\not\cap}, \mathsf{er}{:}\langle\langle \mathsf{book}, \mathsf{bookid} \rangle\rangle \rangle$

$\langle \mathsf{er}{:}\langle\langle \mathsf{paper}, \mathsf{title} \rangle\rangle, \overset{\mathrm{S}}{\not\cap}, \mathsf{er}{:}\langle\langle \mathsf{book}, \mathsf{title} \rangle\rangle \rangle$

$\langle \mathsf{er}{:}\langle\langle \mathsf{paper}, \mathsf{year} \rangle\rangle, \overset{\mathrm{S}}{\not\cap}, \mathsf{er}{:}\langle\langle \mathsf{book}, \mathsf{year} \rangle\rangle \rangle$

$\langle \mathsf{er}{:}\langle\langle \mathsf{author} \rangle\rangle, \overset{\mathrm{S}}{\cap}, \mathsf{er}{:}\langle\langle \mathsf{author} \rangle\rangle \rangle$

$\langle \mathsf{er}{:}\langle\langle \mathsf{author}, \mathsf{name} \rangle\rangle, \overset{\mathrm{S}}{\cap}, \mathsf{er}{:}\langle\langle \mathsf{author}, \mathsf{name} \rangle\rangle \rangle$

$\langle \mathsf{er}{:}\langle\langle \mathsf{writtenby}, \mathsf{paper}, \mathsf{author} \rangle\rangle, \overset{\mathrm{S}}{\not\cap}, \mathsf{er}{:}\langle\langle \mathsf{writtenby}, \mathsf{book}, \mathsf{author} \rangle\rangle \rangle$

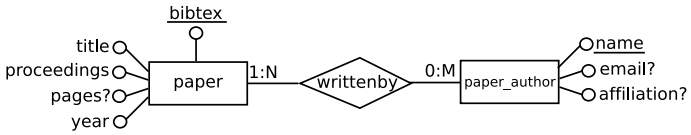
Based on the above mappings, the rules that can be applied in the naming conforming phase are: the Entity Distinction rule due to the mapping $\langle \mathsf{er}{:}\langle\langle \mathsf{author} \rangle\rangle, \overset{\mathrm{S}}{\cap}, \mathsf{er}{:}\langle\langle \mathsf{author} \rangle\rangle \rangle$ and the Attribute Distinction rule due to the mapping $\langle \mathsf{er}{:}\langle\langle \mathsf{author}, \mathsf{name} \rangle\rangle, \overset{\mathrm{S}}{\cap}, \mathsf{er}{:}\langle\langle \mathsf{author}, \mathsf{name} \rangle\rangle \rangle$. In the latter case, the objects have identical names name, they are not equivalent and they are attached to an entity with the same name author.

The Entity Distinction rule generates the following transformations. The transformations are prefixed with the schema they should be executed on.

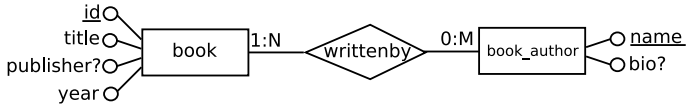
S_1^{er}.renameNode(er:$\langle\langle$author$\rangle\rangle$,er:$\langle\langle$paper_author$\rangle\rangle$)

S_2^{er}.renameNode(er:$\langle\langle$author$\rangle\rangle$,er:$\langle\langle$book_author$\rangle\rangle$)

Regarding the attributes $\mathsf{er}{:}\langle\langle \mathsf{author}, \mathsf{name} \rangle\rangle$ in both schemas, the Edge Dis-



(a) S_1^{er}



(b) S_2^{er}

Figure 6.12: ER schemas $S_1''^{er}$ and $S_2''^{er}$ produced after the naming conforming of S_1^{er} and S_2^{er}

tinction rule would generate similar transformations to rename the objects. However, during the application of the generated transformations, we notice that the conditions of the Attribute Distinction rule do not hold any longer. In particular, due to the rename transformations of the er:⟨⟨author⟩⟩ entities, the attributes are no longer attached to nodes with identical names but have been transformed into er:⟨⟨paper_author, name⟩⟩ and er:⟨⟨book_author, name⟩⟩.

The resulting schemas $S_1''^{er}$ and $S_2''^{er}$ produced after the naming conforming phase are illustrated in Figure 6.12.

◇

The unioning rules for the ER model can also be produced from the generic rules by Exact Translation. For example, the translation of the generic Addition of Union rule into the ER model is performed as follows. The $\texttt{addNodal}_{gen}$ transformation of the generic rule is translated into an addEntity transformation. The next two transformations of the generic rule add a union and an exclusion constraint. These constraints map to a ER generalization constraint, as we have already seen in Section 3.4. Therefore, the next two transformations of the generic rule are translated into a single addGeneralization transformation by exact translation. The final rule, called the **Addition of Union Entity** rule, is the following:

$$\frac{\langle \mathsf{er}{:}\langle\!\langle E_1 \rangle\!\rangle, \overset{\mathrm{S}}{\not\cap}, \mathsf{er}{:}\langle\!\langle E_2 \rangle\!\rangle\rangle \quad \texttt{uniqueName}(E_1, E_2, E', \overset{\mathrm{S}}{\not\cap})}{\begin{array}{l}\mathsf{addEntity}(\mathsf{er}{:}\langle\!\langle E' \rangle\!\rangle, \mathsf{er}{:}\langle\!\langle E_1 \rangle\!\rangle \texttt{ ++ } \mathsf{er}{:}\langle\!\langle E_2 \rangle\!\rangle) \\ \mathsf{addGeneralization}(\mathsf{er}{:}\langle\!\langle E', E_1, E_2 \rangle\!\rangle)\end{array}}$$

Example 6.5. ER Unioning In our running example, in the unioning phase first the schemas $S_1''^{er}$ and $S_2''^{er}$ are extended to produce a single schema where all the objects of both schemas appear and then the Addition of Union Entity and the **Addition of Intersection Entity** rules are applied.

The Addition of Union Entity is applied due to the mapping $\langle \mathsf{er}{:}\langle\!\langle \mathsf{paper} \rangle\!\rangle, \overset{\mathrm{S}}{\not\cap}, \mathsf{er}{:}\langle\!\langle \mathsf{book} \rangle\!\rangle\rangle$. The rule generates the following transformations:

$\mathsf{addEntity}(\mathsf{er}{:}\langle\!\langle \mathsf{paperORbook} \rangle\!\rangle, \mathsf{er}{:}\langle\!\langle \mathsf{paper} \rangle\!\rangle \texttt{ ++ } \mathsf{er}{:}\langle\!\langle \mathsf{book} \rangle\!\rangle)$

$\mathsf{addGeneralization}(\mathsf{er}{:}\langle\!\langle \mathsf{paperORbook}, \mathsf{paper}, \mathsf{book} \rangle\!\rangle)$

The Addition of Intersection Entity rule is applied due to the mapping $\langle \mathsf{er}{:}\langle\!\langle \mathsf{paper\_author} \rangle\!\rangle, \overset{\mathrm{S}}{\cap}, \mathsf{er}{:}\langle\!\langle \mathsf{book\_author} \rangle\!\rangle\rangle$ [3] and generates the transformations:

$\mathsf{addEntity}(\mathsf{er}{:}\langle\!\langle \mathsf{paperANDbookauthor} \rangle\!\rangle, [\{\mathsf{x}\} \mid \{\mathsf{x}\} \leftarrow \mathsf{er}{:}\langle\!\langle \mathsf{paper\_author} \rangle\!\rangle;$

$\{\mathsf{x}\} \leftarrow \mathsf{er}{:}\langle\!\langle \mathsf{book\_author} \rangle\!\rangle])$

$\mathsf{addSubset}(\mathsf{er}{:}\langle\!\langle \mathsf{paper\_author}, \mathsf{paperANDbookauthor} \rangle\!\rangle)$

$\mathsf{addSubset}(\mathsf{er}{:}\langle\!\langle \mathsf{book\_author}, \mathsf{paperANDbookauthor} \rangle\!\rangle)$

The final schema S_{12}^{er} produced in the unioning phase is illustrated in Figure 6.13.

◇

Finally in the restructuring phase, Meta-Constraint Restriction cases can be identified when translating from the generic rules to ER model rules.

[3] The $\mathsf{er}{:}\langle\!\langle \mathsf{author} \rangle\!\rangle$ entities have been renamed during the naming conforming phase

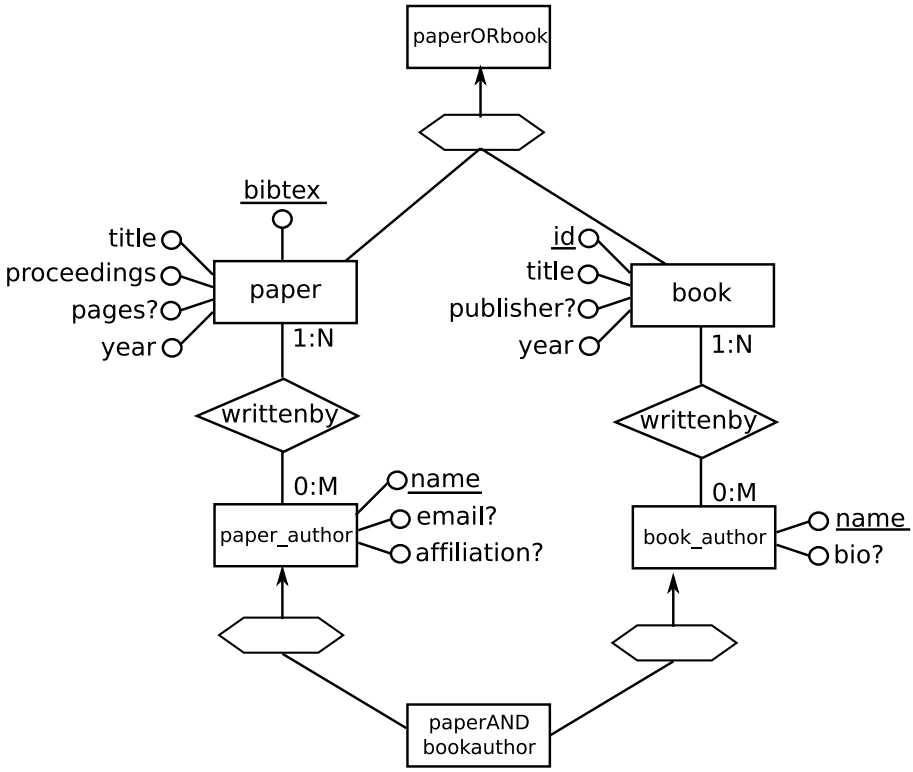


Figure 6.13: Schema S_{12}^{er} produced after unioning schemas $S_1''^{er}$ and $S_2''^{er}$.

For example, consider the Generalization of Link-Nodals rule in Figure 6.9, which is translated into the **Generalization of Attributes** rule in the ER model and applied for two disjoint attributes $\mathsf{er}{:}\langle\langle X_1, N_1, C\rangle\rangle$ and $\mathsf{er}{:}\langle\langle X_2, N_2, C\rangle\rangle$. The $\texttt{addLN}_{gen}$ operation of the generic rule can be redefined in the ER model using Exact Translation into a $\texttt{addLN}_{er}$ operation. The $\texttt{addLN}_{er}$ operation before performing the corresponding high level transformation, *i.e.* $\mathsf{addAttribute}(\mathsf{er}{:}\langle\langle X, N, C\rangle\rangle)$, identifies the common constraints of the existing attributes and then cascades them into the new attribute. The constraints that are relevant to an ER attributes are whether the attributes are nullable or not, and whether the attributes are keys. The less restrictive case is adopted by the transformation.

If the attributes $\mathsf{er}{:}\langle\langle X_1, N_1, C\rangle\rangle$ and $\mathsf{er}{:}\langle\langle X_2, N_2, C\rangle\rangle$ have identical nullability constraints C, *i.e.* both attributes are either nullable or non-nullable, the generalized attribute will adopt the same constraints C. In addition, if both attributes are keys on the table they are attached to, checked using the auxiliary `key/1` function, then

the added attribute also becomes a key:

$$\texttt{addLN}_{er}(\texttt{er:}\langle\langle X, N\rangle\rangle, Q, \texttt{er:}\langle\langle X_1, N_1, C\rangle\rangle, \texttt{er:}\langle\langle X_2, N_2, C\rangle\rangle) \text{ :-}$$
$$\texttt{addAttribute}(\texttt{er:}\langle\langle X, N, C\rangle\rangle, Q),$$
$$(\texttt{key}(\texttt{er:}\langle\langle X_1, N_1\rangle\rangle), \texttt{key}(\texttt{er:}\langle\langle X_2, N_2\rangle\rangle, \texttt{addKey}(\texttt{er:}\langle\langle X, N\rangle\rangle))).$$

If the attributes do not agree on their constraints, $C_1 \neq C_2$, *i.e.* one is nullable and the other non-nullable, the generalized attribute will be nullable, which is less restrictive. Notice that in this case, we already know that one of the attributes cannot be a key, in particular the nullable attribute, whichever that is:

$$\texttt{addLN}_{er}(\texttt{er:}\langle\langle X, N\rangle\rangle, Q, \texttt{er:}\langle\langle X_1, N_1, C_1\rangle\rangle, \texttt{er:}\langle\langle X_2, N_2, C_2\rangle\rangle) \text{ :-}$$
$$C_1 \neq C_2, \texttt{ addAttribute}(\texttt{er:}\langle\langle X, N, null\rangle\rangle, Q).$$

The $\texttt{deleteLN}_{gen}$ function in the Generalization of Link-Nodals rule can be redefined using a Meta-Constraint Restriction. In the ER model, the attribute cannot be deleted if it is a key because it identifies each instance of the entity:

$$\texttt{deleteLN}_{er}(\texttt{er:}\langle\langle X, N, C\rangle\rangle, Q) \text{ :-}$$
$$\neg\ \texttt{key}(\texttt{er:}\langle\langle X, N\rangle\rangle), \texttt{ deleteAttribute}(\texttt{er:}\langle\langle X, N, C\rangle\rangle, Q).$$

Example 6.6. Restructuring: Generalization of Attributes

In our running example, in the restructuring phase the Generalization of Attributes can be applied on schema S_{12}^{er} due to the mapping $\langle \texttt{er:}\langle\langle \textsf{paper}, \textsf{bibtex}\rangle\rangle, \overset{\text{S}}{\eta}, \texttt{er:}\langle\langle \textsf{book}, \textsf{id}\rangle\rangle\rangle$. The two attributes are generalized into $\texttt{er:}\langle\langle \textsf{paperORbook}, \textsf{id}, \textsf{notnull}\rangle\rangle$, which also becomes the key of the table $\texttt{er:}\langle\langle \textsf{paperORbook}\rangle\rangle$. The $\texttt{er:}\langle\langle \textsf{paper}, \textsf{bibtex}\rangle\rangle$ and $\texttt{er:}\langle\langle \textsf{book}, \textsf{id}\rangle\rangle$ attributes are not deleted because they are keys. The rule generates the following transformations:

$$\textsf{addAttribute}(\texttt{er:}\langle\langle \textsf{paperORbook}, \textsf{id}, \textsf{notnull}\rangle\rangle,$$
$$\texttt{er:}\langle\langle \textsf{paper}, \textsf{bibtex}\rangle\rangle \text{ ++ } \texttt{er:}\langle\langle \textsf{book}, \textsf{id}\rangle\rangle)$$
$$\textsf{addKey}(\texttt{er:}\langle\langle \textsf{paperORbook}, \textsf{id}\rangle\rangle)$$

Additionally, the Generalization of Attributes rule can be applied due to the mapping $\langle \texttt{er:}\langle\langle \textsf{paper}, \textsf{year}\rangle\rangle, \overset{\text{S}}{\eta}, \texttt{er:}\langle\langle \textsf{book}, \textsf{year}\rangle\rangle\rangle$. The two attributes are

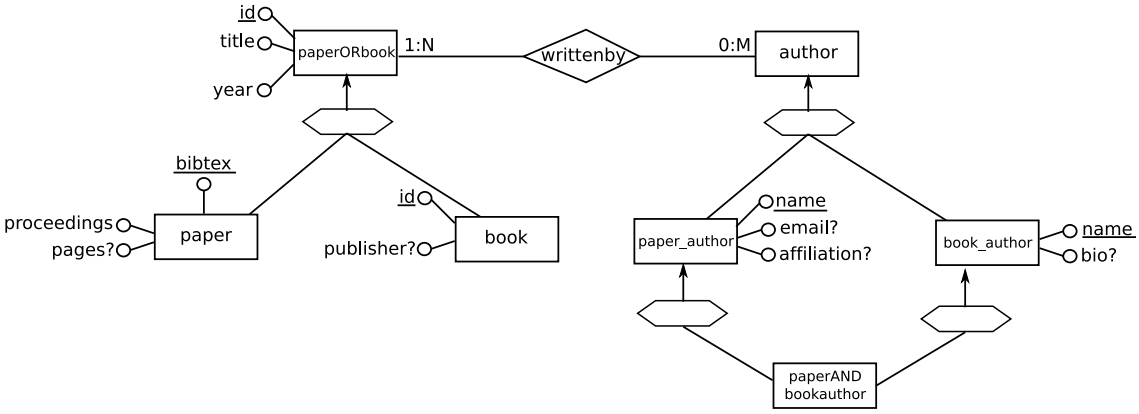


Figure 6.14: The final result of the integration of schemas S_1^{er} and S_2^{er}

generalized into er:⟨⟨paperORbook, year, notnull⟩⟩ and then they are deleted since they are not key attributes:

addAttribute(er:⟨⟨paperORbook, year, notnull⟩⟩,
er:⟨⟨paper, year⟩⟩] ++ er:⟨⟨book, year⟩⟩)
deleteAttribute(er:⟨⟨paper, year⟩⟩,
[{x, y} | {x, y} ← er:⟨⟨paperORbook, year⟩⟩; {x} ← er:⟨⟨paper⟩⟩]
deleteAttribute(er:⟨⟨book, year⟩⟩,
[{x, y} | {x, y} ← er:⟨⟨paperORbook, year⟩⟩; {x} ← er:⟨⟨book⟩⟩]

Further restructuring rules, including the **Generalization of ER Relationships** rule, produce the final integrated ER schema, which is illustrated in Figure 6.14. Comparing this schema with the final integrated HDM schema of S_1 and S_2, illustrated in Figure 6.7, shows that the ER schema is almost the exact translation of the HDM schema into the ER model. However, the HDM schema also includes some additional nodes produced due to the lack of distinction between nodes that represent nodal objects and nodes of link-nodal constructs.

◇

6.4 Top-K Schema Merging

In the previous sections, we introduced our schema merging process and showed how each integrated schema is produced based on a single schema mapping. In this section, we give an example of the final outcome of our top-K schema integration approach.

In our top-K schema integration approach, there are top-K schema mappings (Definition 4.11) produced in the matching process. Each schema mapping sm is associated with a degree of belief or plausibility l, derived from the matching process. For each sm_i with belief or plausibility l_i, we perform the schema merging process presented in the previous sections and produce an integrated schema S_i. It could be the case that for distinct schema mappings sm_i, sm_j, $i \neq j$, an identical integrated schema S_{ij} is produced. The belief or plausibility of each schema S_i is equivalent to the normalized sum of the beliefs or plausibilities of the schema mappings sm_i schema S_i has been derived from. Thus, the final outcome of our schema integration approach is a list of at most K integrated schemas, ranked based on their belief or plausibility, which we call the top-K integrated schemas (Definition 4.12).

Example 6.7. Top-K schema merging

In this example, we are going to merge the ER schemas S_1^{er} and S_2^{er} presented in the introduction and illustrated in Figure 6.11. We are going to merge the two schemas based on the uncertain semantic mappings (USMs) between pairs

$$p_1 \equiv [\text{er:}\langle\langle \mathsf{paper} \rangle\rangle, \text{er:}\langle\langle \mathsf{book} \rangle\rangle]$$

$$p_2 \equiv [\text{er:}\langle\langle \mathsf{paper}, \mathsf{bibtex} \rangle\rangle, \text{er:}\langle\langle \mathsf{book}, \mathsf{id} \rangle\rangle]$$

$$p_3 \equiv [\text{er:}\langle\langle \mathsf{paper}, \mathsf{title} \rangle\rangle, \text{er:}\langle\langle \mathsf{book}, \mathsf{title} \rangle\rangle]$$

$$p_4 \equiv [\text{er:}\langle\langle \mathsf{paper}, \mathsf{year} \rangle\rangle, \text{er:}\langle\langle \mathsf{book}, \mathsf{year} \rangle\rangle]$$

$$p_5 \equiv [\text{er:}\langle\langle \mathsf{author} \rangle\rangle, \text{er:}\langle\langle \mathsf{author} \rangle\rangle]$$

$$p_6 \equiv [\text{er:}\langle\langle \mathsf{author}, \mathsf{name} \rangle\rangle, \text{er:}\langle\langle \mathsf{author}, \mathsf{name} \rangle\rangle]$$

The USMs are illustrated in Figure 1.4 and they are formally listed below:

$$m_1(\{\overset{\text{S}}{\not\sim}\}) = .65, m_1(\{\overset{\text{S}}{\not\equiv}\}) = .35$$

$$\langle \text{er:}\langle\langle \mathsf{paper} \rangle\rangle, m_1, \text{er:}\langle\langle \mathsf{book} \rangle\rangle \rangle$$

$$\langle \text{er:}\langle\langle \mathsf{paper}, \mathsf{bibtex} \rangle\rangle, m_1, \text{er:}\langle\langle \mathsf{book}, \mathsf{id} \rangle\rangle \rangle$$

$$m_2(\{\overset{\text{S}}{\not\equiv}\}) = .90, m_2(\{\overset{\text{S}}{\not\sim}\}) = .10$$

$$\langle \text{er:}\langle\langle \mathsf{paper}, \mathsf{title} \rangle\rangle, m_2, \text{er:}\langle\langle \mathsf{book}, \mathsf{title} \rangle\rangle \rangle$$

$$\langle \text{er:}\langle\langle \mathsf{paper}, \mathsf{year} \rangle\rangle, m_2, \text{er:}\langle\langle \mathsf{book}, \mathsf{year} \rangle\rangle \rangle$$

$$m_3(\{\overset{\text{S}}{=}\}) = 1.0$$

$$\langle \text{er:}\langle\langle \mathsf{author} \rangle\rangle, m_3, \text{er:}\langle\langle \mathsf{author} \rangle\rangle \rangle$$

$$\langle \text{er:}\langle\langle \mathsf{author}, \mathsf{name} \rangle\rangle, m_3, \text{er:}\langle\langle \mathsf{book}, \mathsf{name} \rangle\rangle \rangle$$

$$m_4(\{\overset{\text{S}}{\not\equiv}\}) = 1.0$$

$$\langle \text{er:}\langle\langle \mathsf{writtenby}, \mathsf{paper}, \mathsf{author} \rangle\rangle, m_3, \text{er:}\langle\langle \mathsf{writtenby}, \mathsf{paper}, \mathsf{author} \rangle\rangle \rangle$$

All the remaining pairs of objects between the two schemas are incompatible.

Notice that all the BPAs (Definition 4.7) above, m_1, m_2, m_3 and m_4, assign probability mass to singleton semantic relationships. Thus, the belief and plausibility for each of these semantic relationships is equivalent to its probability mass (Definition 4.8). For example, for the pair of objects

sm	p_1	p_2	p_3	p_4	p_5	p_6	p_7	Belief/Plausibility
#1	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.3422
#2	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.1843
#3	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.1843
#4	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0992
#5	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0380
#6	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0380
#7	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0204
#8	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0204
#9	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0204
#11	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0204
#11	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0110
#12	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0110
#13	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0042
#14	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0022
#15	$\overset{s}{\nsim}$	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0022
#16	$\overset{s}{\not\sim}$	$\overset{s}{\not\sim}$	$\overset{s}{\nsim}$	$\overset{s}{\nsim}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{\not\sim}$	0.0012

Table 6.2: 16 possible schema mappings for the integration of S_1^{er} and S_2^{er}

er:$\langle\langle$paper$\rangle\rangle$, er:$\langle\langle$book$\rangle\rangle$ we have

$$Bel(\{\overset{s}{\nsim}\}) = \sum_{A\subseteq\{\overset{s}{\nsim}\}.A\in\Theta_{rel}} m_1(A) = m_1(\{\overset{s}{\nsim}\}) = 0.65$$

Based on the above USMs, there are two possible semantic relationships for pairs p_1, p_2, and two possible semantic relationships for pairs p_3, p_4. Thus, the USMs specify $(2 \times 2) \times (2 \times 2) = 16$ possible schema mappings in total. The semantic relationship for each pair of objects in these 16 schema mappings is illustrated in Table 6.2. The belief/plausibility of each schema mapping in the table is computed as the product of the beliefs/plausibilities of the semantic relationships for each pair the schema mapping is composed of. For example, to compute the belief/plausibility of schema mapping #1 we multiply the beliefs/plausibilities of each relationship for each pair: $0.65.65 \times 0.9 \times 0.9 \times 1 \times 1 \times 1 = 0.3422$.

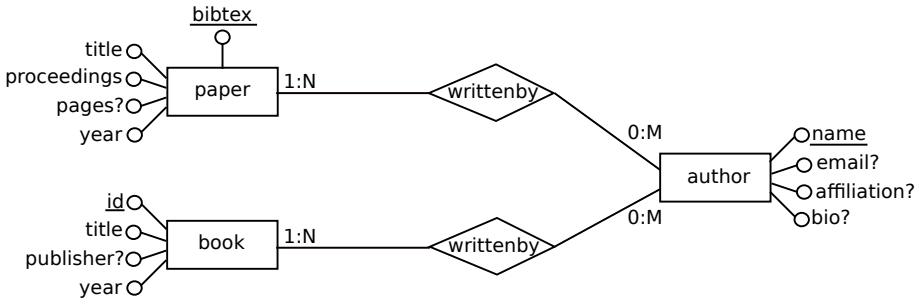


Figure 6.15: Schema S_1 produced based on schema mappings #1 and #3

Assume that we are interested in the top-3 schema mappings, *i.e.* schema mappings #1, #2 and #3. For each schema mapping, we are going to apply ER merging rules based on the generic merging rules in Section 6.3.

sm #1 : First, we investigate the naming conforming rules and we identify that there are no rules that can be applied. For example, *sm* #1 specifies that the attributes in p_3, $\mathsf{er}{:}\langle\langle \mathsf{paper}, \mathsf{title} \rangle\rangle$ and $\mathsf{er}{:}\langle\langle \mathsf{book}, \mathsf{title} \rangle\rangle$, are equivalent. However, the Attribute Distinction rule (Figure 6.10(b)) is not satisfied and does not have to be applied because the attributes are of entities with distinct names, paper and book, *i.e.* the first condition of the rule is violated. Then, we investigate unioning rules, but none of them are applied since there are no subsumption, intersection nor disjointness relationships between any ER entities. Finally, no restructuring rules can be applied even though there are disjoint ER attributes (pairs p_3 and p_4) and a disjoint ER relationship (pair p_7), because their associated ER entities are not disjoint as the rules require, *e.g.* the Generalization of Link-Nodals rule in Figure 6.9. Thus, the integrated schema for *sm* #1 is produced by superimposing the two schemas. The final schema S_1 is illustrated in Figure 6.15.

sm #2 : This schema mapping is different from *sm* #1 only on the semantic relationship for p_1. During merging, again there are no rules that can be applied during naming conforming. In the unioning phase,

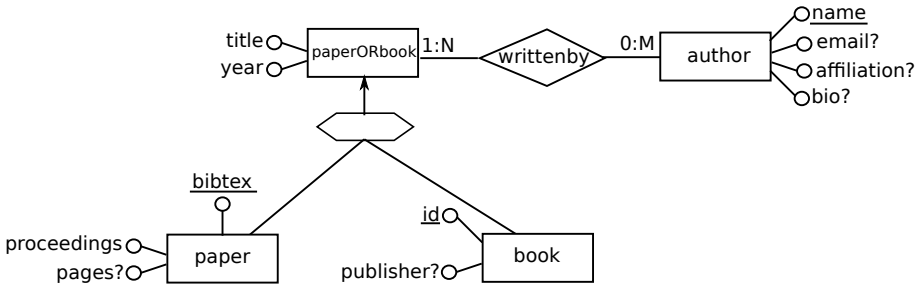


Figure 6.16: Schema S_2 produced based on schema mapping #2

the disjoint ER entities of p_1 allow the application of the Addition of Union Entity rule, presented in Section 6.3, and a new ER entity is added er:⟨⟨paperORbook⟩⟩. Finally, in the restructuring phase, as in Example 6.6, we can apply both the Generalization of Attributes rule for the disjoint pairs of attributes p_3 and p_4 and the Generalization of ER Relationships rule for the disjoint ER relationships of pair p_7. The final schema S_2 is illustrated in Figure 6.16.

sm #3 : This schema mapping is different from *sm* #1 only on the semantic relationship for p_2. Pair p_2 is a pair of attributes which are considered in *sm* #3 to be disjoint and do not have identical names. Therefore, as in *sm* #1, there are no naming conforming rules to be applied, nor unioning and restructuring rules. Thus, the integrated schema for *sm* #3 is schema S_1, as well.

Thus, our top-K schema merging approach produces two schemas S_1 and S_2. The belief of these two schemas is the normalized sum of the beliefs of the schema mappings the schemas are derived from. The total belief of schema mappings *sm* #1, *sm* #2 and *sm* #3 is $0.3422 + 0.1843 + 0.1843 = 0.7108$. Therefore, schema S_1, which is derived from schema mappings #1 and #3, is assigned belief $\frac{0.3422+0.1843}{0.7108} = 0.74$, and schema S_2, which is derived from schema mapping #2, is assigned belief $\frac{0.1843}{0.7108} = 0.26$. The final result of merging S_1^{er} and S_2^{er} is a list of two integrated schemas (together with their BAV view definitions produced by the application of the merging rules)

annotated with a degree of belief: $[(S_1, 0.74), (S_2, 0.26)]$.

$\diamond$

6.5 Summary

In this chapter, we have presented our approach to schema merging.

To avoid the complexity of high-level data models, we introduce our merging approach on the low-level HDM. We examine exhaustively each possible schema mapping between a pair of objects and each possible sub-schema configuration. Each case identified is used to specify a low-level merging rule. Our merging rules are formally defined and they are precise, using BAV transformations to integrate the input schemas.

We improve existing approaches (reviewed in Section 2.3.3) by either

- producing view definitions between the input and integrated schemas: There are existing approaches in the literature [59, 101, 89] that define an integrated schema based on semantic mappings (equivalence, subsumption, *etc*). However, these approaches do not specify view definitions between the input and integrated schemas. Without view definitions, data cannot be queried using the integrated schema.

- or by using semantic mappings (equivalence, subsumption, *etc*) instead of data mappings: There are existing approaches in the literature [87, 74, 12] that define view definitions between the input and the integrated schemas, but these approaches are based on given data mappings between the input schema objects. These data mappings are essentially arbitrary queries between the input schema objects, therefore a lot of this work is related to reasoning about queries, which we do not do in our approach. Instead, we perform schema

merging based on semantic mappings, which are more high-level correspondences between schema objects. In addition, semantic mappings are most commonly identified by existing schema matching approaches (Table 2.1), while the identification of data mappings is a widely accepted hard problem [31].

Another advantage of our approach is that it allows the existence of a generic schema merging framework, which is model independent and which can be translated to high-level data models. In this chapter, we presented the methodology to derive the generic schema merging framework and how it can be used to produce ER merging rules. In addition, our proposed framework allows us to reason about the extent our merging rules can be automated, and the soundness and completeness of the view definitions we create.

The final outcome of our top-K schema merging is a list of at most K integrated schemas each one associated with a degree of uncertainty, which we call top-K integrated schemas.

Chapter 7

Conclusions and Future Work

In this dissertation, we have presented our schema integration framework based on uncertain semantic mappings. Our framework represents and manages the inherent uncertainty of the schema matching process and provides a low-level approach on schema merging extensible for high-level schemas. In this final chapter of this dissertation, we give our final conclusions on the research we conducted.

The objective in schema integration is the combination of data from different data sources by creating a unified schema of the data. To achieve this, there are two main tasks: **schema matching** and **schema merging**. In schema matching, a **schema mapping** is identified consisting of a list of semantic mappings between the input schema objects, while schema merging uses the identified schema mapping to produce the final **integrated schema**.

It is widely accepted that schema matching is a very hard task and that in general there is no unique solution to the problem. Even though researchers agree that automatic schema matching software are error prone, there are few attempts in the literature that consider providing feedback to the user about the correctness of the schema mapping discovered during matching.

In our approach, we have attempted to capture the correctness of the schema map-

ping discovered by automatic schema matching software by representing the uncertainty of the matching tool about the semantic mappings between schema objects. Our approach introduces the notion of **uncertain semantic mapping** for schema objects. Uncertain semantic mappings can be used to specify the levels of certainty of correctness for the possible semantic relationships between the objects. For example, a matching tool could report that it is certain that two objects s_1 and s_2 are not equivalent, but it is 40% certain that s_1 subsumes s_2, 30% certain that s_1 and s_2 are intersecting, 20% certain that they are disjoint and 10% that they are incompatible.

The uncertain semantic mappings discovered during matching define an **uncertain schema mapping**, which allows several different integrations of the input schemas. In fact, the number of possible integrations is exponential. Thus, we are only interested in identifying the most certain integrated schemas, *i.e.* the **top-K integrated schemas**.

The decision on whether a top-1 or a top-K, K$>$ 1, schema integration approach is used depends on the objectives of the user and the application setting. In a fully automated environment, the top-K approach should be used since we show in this dissertation that it produces schema mappings with higher accuracy than the top-1 approach. In a semi-automated environment, we have identified two user objectives. If the user wants to identify *the* single correct schema mapping, then the top-1 approach should be used. The user can then examine and correct either the schema mapping discovered or the resulting integrated schema. If the user finds the manual cost of correcting the schema mapping prohibiting, then the top-K approach should be used. In our top-K approach, the pairs of objects which the tool is least certain about their mappings are directly available to the user. Thus, the user can only correct the mappings of these pairs, which is more cost effective than correcting the whole schema mapping. Additionally, the user can go through the top-K schema mappings and select the one she prefers.

Regarding the top-K integrated schema, our schema merging process generates each integrated schema based on a list of semantic mappings identified during schema matching. Our merging process is based on a set of formal low-level rules, which we have identified by exhaustively investigating each possible semantic mapping between a pair of objects in each possible sub-schema configuration. Each low-level rule specifies both the structure of the integrated schema and **view definitions** between the input and the integrated schemas.

In this dissertation, we show that based on the low-level merging rules we can produce a **generic** schema merging framework, which we can extend to support different high-level data models. Thus, our merging approach can be used to integrate schemas of both low-level and high-level data models.

Due to the formal definition of the merging rules, we are able to prove that our merging process does not cause any information loss nor gain. Thus, the final integrated schema produced is both **sound** and **complete**.

7.1 Comparison to Related Work

Our top-K schema integration framework subsumes existing approaches [20, 83, 62, 35, 9, 2, 38, 65, 33, 73, 10, 34, 47, 112] that do not take into consideration the uncertainty of schema matching and produce one schema mapping for each matching task, and thus a single integrated schema is produced. We can simulate these approaches by identifying the top-1 schema mapping and the top-1 integrated schemas. The experimental evaluation of our prototype implementation shows that using our approach improves the accuracy of existing matching algorithms [104, 75, 73] even for the identification of the top-1 schema mapping.

As far as we know, there are only two other existing approaches [46, 81] that take into account the uncertainty during schema matching. Our approach subsumes both

approaches which can only be used for identifying compatibility mappings, while our framework supports, in addition to compatibility, five more precise semantic mappings. Additionally, in both [46] and [81] the final result of the schema matching process does not depict the uncertainty of the matching tool but rather the confidence assigned by the user to each matching algorithm.

As far as we know, there is no other schema merging approach that uses as input uncertain mappings between schemas. In our merging approach, an uncertain schema mapping results into top-K integrated schemas, which are materialized and each one is assigned a level of uncertainty. In [37], which is the work most related to this problem, the authors do not deal with schema merging *per se* but instead investigate answering queries based on uncertain mappings.

Regarding the production of each individual integrated schema, our approach improves against existing approaches [87, 74, 12] that only define the structure of the integrated schema. Our approach also generates view definitions between the input and integrated schemas. Compared to [31], which also generates view definitions, our approach is more general since it provides a generic framework for the integration of schemas of any high-level data model, while in [31] only the relational data model is considered.

7.2 Future Work

In the future, we could work on both improving our prototype implementation and examining research problems that our approach could be applied on.

Regarding our prototype implementation, we could improve our matching experts by investigating more sophisticated training processes for the derivation of uncertain semantic mappings. This could potentially improve the accuracy of the schema mappings and the integrated schemas our tool produces. Additionally, we could

incorporate further schema matching algorithms to both improve the accuracy of our tool and empirically prove that our approach improves the results these algorithms already produce.

For a better evaluation of our approach, it would be useful to produce and examine test cases with more precise semantic mappings between schemas. Our current data set taken from [73] only considered compatibility mappings and therefore we were not able to evaluate the full potential of our framework, which can deal with six types of semantic mappings.

It would also be very interesting to investigate whether our approach can be extended for the identification of data mappings. Currently, we identify uncertain semantic mappings. It would be interesting to see how uncertainty can be represented on data mappings, such as "name equals to the concatenation of first-name and last-name".

Regarding future research directions, we have identified that lately the issue of uncertainty in schema integration has been considered in several research papers [31, 49, 66]. One area where uncertainty is inherent and where our research would have direct application is dataspaces [49]. Dataspaces have been proposed as a data management abstraction in settings where there is an increasing number of diverse, interrelated data sources with no means of managing them in a convenient principled way. In these settings, mappings are approximate, queries are not structured and data are imprecise. Even in this situation, there is a need for a basic functionality over all data sources regardless how integrated they are, *e.g.* supporting keyword queries without the existence of a single integrated schema. When more sophisticated operations are required, schema integration can be performed locally in an incremental pay-as-you-go fashion. In these cases, some cost needs to be paid for better integration results and better query answers.

Our work, which allows the automatic identification of top-K integrated schemas, is directly applicable in dataspaces, since integration can be performed at no user cost.

If more accurate results are required, then the process described in this dissertation can be used, where users examine the top-K integrated schemas and select the ones they prefer. In this case, the users have to pay a small cost to improve the integration and thus query results.

Another research area we could apply our approach on is model management [11]. In model management, schemas are treated as bulk objects using high-level operators such as **Match**, **Merge**, **Compose**, *etc.* We could investigate how these operators can be extended to support uncertainty. Our research has effectively already extended the **Match** operator, which performs schema matching, and the **Merge** operator, which performs schema merging. Additionally, we could investigate the remaining operators. For example, the **Compose** operator combines certain mappings between schemas S_1 and S_2, and S_2 and S_3, to produce a mapping between S_1 and S_3. It would be interesting to see what is the meaning of combining uncertain mappings and how uncertain mappings *can* be combined.

Appendix A

Low-Level Schema Merging Rules

In this appendix we list all low-level schema merging rules.

A.1 Naming Conforming

The naming conforming rules are:

$\langle \langle\!\langle N_1 \rangle\!\rangle, \stackrel{\mathrm{S}}{=}, \langle\!\langle N_2 \rangle\!\rangle \rangle$	$\neg\ \langle \langle\!\langle N_1 \rangle\!\rangle, \stackrel{\mathrm{S}}{=}, \langle\!\langle N_2 \rangle\!\rangle \rangle$
$\neg$ `identicalNames`(N_1, N_2)	`identicalNames`(N_1, N_2)
`commonName`(N_1, N_2, N')	`distinctNames`(N_1, N_2, N_1', N_2')
renameNode$(\langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N' \rangle\!\rangle)$	renameNode$(\langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N_1' \rangle\!\rangle)$
renameNode$(\langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle N' \rangle\!\rangle)$	renameNode$(\langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle N_2' \rangle\!\rangle)$
$\langle\!\langle e_1, N_1, N_2 \rangle\!\rangle \stackrel{\mathrm{S}}{=} \langle\!\langle e_2, N_1, N_2 \rangle\!\rangle$	$\neg\ \langle\!\langle e_1, N_1, N_2 \rangle\!\rangle \stackrel{\mathrm{S}}{=} \langle\!\langle e_2, N_1, N_2 \rangle\!\rangle$
$\neg$ `identicalNames`(e_1, e_2)	`identicalNames`(e_1, e_2)
`commonName`(e_1, e_2, e')	`distinctNames`(e_1, e_2, e_1', e_2')
renameEdge$(\langle\!\langle e_1, N_1, X_1 \rangle\!\rangle, \langle\!\langle e', N_1, X_1 \rangle\!\rangle)$	renameEdge$(\langle\!\langle e_1, N_1, N_2 \rangle\!\rangle, \langle\!\langle e_1', N_1, N_2 \rangle\!\rangle)$
renameEdge$(\langle\!\langle e_2, N_2, X_2 \rangle\!\rangle, \langle\!\langle e', N_2, X_2 \rangle\!\rangle)$	renameEdge$(\langle\!\langle e_2, N_1, N_2 \rangle\!\rangle, \langle\!\langle e_2', N_1, N_2 \rangle\!\rangle)$

A.2 Unioning

The unioning rules are:

	$\langle\langle\langle N_1\rangle\rangle, \overset{S}{\cap}, \langle\langle N_2\rangle\rangle\rangle$
	uniqueName(N_1,N_2,N',$\overset{S}{\cap}$)
$\langle\langle\langle N_2\rangle\rangle, \overset{S}{\subset}, \langle\langle N_1\rangle\rangle\rangle$	addNode($\langle\langle N'\rangle\rangle$, [{$x$} | {$x$} $\leftarrow$ $\langle\langle N_1\rangle\rangle$;
addConstraint($\langle\langle \subseteq, \langle\langle N_2\rangle\rangle, \langle\langle N_1\rangle\rangle\rangle\rangle$)	{x} $\leftarrow$ $\langle\langle N_2\rangle\rangle$])
	addConstraint($\langle\langle \subseteq, \langle\langle N'\rangle\rangle, \langle\langle N_1\rangle\rangle\rangle\rangle$)
	addConstraint($\langle\langle \subseteq, \langle\langle N'\rangle\rangle, \langle\langle N_2\rangle\rangle\rangle\rangle$)

$\langle\langle\langle N_1\rangle\rangle, \overset{S}{\not\cap}, \langle\langle N_2\rangle\rangle\rangle$
uniqueName(N_1,N_2,N',$\overset{S}{\not\cap}$)
addNode($\langle\langle N'\rangle\rangle$, $\langle\langle N_1\rangle\rangle$ ++ $\langle\langle N_2\rangle\rangle$)
addConstraint($\langle\langle \cup, \langle\langle N'\rangle\rangle, \langle\langle N_1\rangle\rangle, \langle\langle N_2\rangle\rangle\rangle\rangle$)
addConstraint($\langle\langle \not\cap, \langle\langle N_1\rangle\rangle, \langle\langle N_2\rangle\rangle\rangle\rangle$)

A.3 Restructuring

The Redundant Edge Removal rules are:

	$\langle\langle e_1, N_1, N_1'\rangle\rangle \overset{S}{=} \langle\langle e_2, N_2, N_2'\rangle\rangle$
$\langle\langle e_1, N_{1/2}, N_1'\rangle\rangle \overset{S}{=} \langle\langle e_2, N_{1/2}, N_2'\rangle\rangle$	$\langle\langle N_2\rangle\rangle \overset{S}{\subset} \langle\langle N_1\rangle\rangle$
$\langle\langle N_2'\rangle\rangle \overset{S}{\subset} \langle\langle N_1'\rangle\rangle$	$\langle\langle N_2'\rangle\rangle \overset{S}{\subset} \langle\langle N_1'\rangle\rangle$
constraints($\langle\langle N_1'\rangle\rangle$, $\langle\langle e_1, N_{1/2}, N_1'\rangle\rangle$,	constraints($\langle\langle N_1\rangle\rangle$, $\langle\langle e_1, N_1, N_1'\rangle\rangle$,
$Constraints$)	$Constraints_1$)
	constraints($\langle\langle N_1'\rangle\rangle$, $\langle\langle e_1, N_1, N_1'\rangle\rangle$,
	$Constraints_1'$)
genDeleteCons($Constraints$)	genDeleteCons($Constraints_1$)
	genDeleteCons($Constraints_1'$)
moveDependents(e_1, e_2)	moveDependents(e_1, e_2)
deleteEdge($\langle\langle e_1, N_{1/2}, N_1'\rangle\rangle$, [{$x, y$} |	deleteEdge($\langle\langle e_1, N_1, N_1'\rangle\rangle, \langle\langle e_2, N_2, N_2'\rangle\rangle$)
{x, y} $\leftarrow$ $\langle\langle e_2, N_{1/2}, N_2'\rangle\rangle$])	

$\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle \overset{\mathrm{S}}{=} \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle$

$\langle\!\langle N_2 \rangle\!\rangle \overset{\mathrm{S}}{\subset} \langle\!\langle N_1 \rangle\!\rangle$

$\langle\!\langle N_1' \rangle\!\rangle \overset{\mathrm{S}}{\subset} \langle\!\langle N_2' \rangle\!\rangle$

`constraints(`$\langle\!\langle N_1 \rangle\!\rangle$`,`$\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle$`,`$ConstraintsE1$`)`

`constraints(`$\langle\!\langle N_2' \rangle\!\rangle$`,`$\langle\!\langle e_2, N_2, N_2' \rangle\!\rangle$`,`$ConstraintsE2$`)`

`uniqueName(`$e_1, e_2, e', \overset{\mathrm{S}}{=}$`)`

---

addEdge($\langle\!\langle e', N_2, N_1' \rangle\!\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle]$)

`genDeleteCons(`$ConstraintsE1$`)`

`moveDependents(`e_1`,`e'`)`

deleteEdge($\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e', N_2, N_1' \rangle\!\rangle]$)

`genDeleteCons(`$ConstraintsE2$`)`

`moveDependents(`e_2`,`e'`)`

deleteEdge($\langle\!\langle e_2, N_2, N_2' \rangle\!\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e', N_2, N_1' \rangle\!\rangle]$)

The Specialization of Edges rules are:

	$\langle\langle e_1, N_1, N_1' \rangle\rangle \stackrel{S}{=} \langle\langle e_2, N_2, N_2' \rangle\rangle$
	$\langle\langle N_2 \rangle\rangle \stackrel{S}{\subset} \langle\langle N_1 \rangle\rangle$
$\langle\langle e_1, N_{1/2}, N_1' \rangle\rangle \stackrel{S}{=} \langle\langle e_2, N_{1/2}, N_2' \rangle\rangle$	$\langle\langle N_1' \rangle\rangle \stackrel{S}{\cap} \langle\langle N_2' \rangle\rangle$
$\langle\langle N_1' \rangle\rangle \stackrel{S}{\cap} \langle\langle N_2' \rangle\rangle$	createdNodal($\langle\langle N_1' \rangle\rangle, \langle\langle N_2' \rangle\rangle, \langle\langle N' \rangle\rangle$)
createdNodal($\langle\langle N_1' \rangle\rangle, \langle\langle N_2' \rangle\rangle, \langle\langle N' \rangle\rangle$)	uniqueName($e_1, e_2, e', \stackrel{S}{=}$)
uniqueName($e_1, e_2, e', \stackrel{S}{=}$)	constraints($\langle\langle N_1 \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle$, $Constraints_1$)
constraints($\langle\langle N_1' \rangle\rangle, \langle\langle e_1, N_{1/2}, N_1' \rangle\rangle$, $Constraints_1$)	constraints($\langle\langle N_1' \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle$, $Constraints_1'$)
constraints($\langle\langle N_2' \rangle\rangle, \langle\langle e_2, N_{1/2}, N_2' \rangle\rangle$, $Constraints_2$)	constraints($\langle\langle N_2' \rangle\rangle, \langle\langle e_2, N_2, N_2' \rangle\rangle$, $Constraints_2'$)
commonCons($Constraints_1, Constraints_2$, $Constraints$)	commonCons($Constraints_1'$, $Constraints_2', Constraints$)
addEdge($\langle\langle e', N_{1/2}, N' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e_2, N_{1/2}, N_2' \rangle\rangle]$)	addEdge($\langle\langle e', N_2, N' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e_2, N_2, N_2' \rangle\rangle]$)
addConsList($Constraints, \langle\langle N' \rangle\rangle$, $\langle\langle e', N_{1/2}, N' \rangle\rangle$)	addConsList($Constraints, \langle\langle N' \rangle\rangle$, $\langle\langle e', N_2, N' \rangle\rangle$)
genDeleteCons($Constraints_1$)	genDeleteCons($Constraints_1$)
moveDependents(e_1, e')	genDeleteCons($Constraints_1'$)
deleteEdge($\langle\langle e_1, N_{1/2}, N_1' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N_{1/2}, N' \rangle\rangle]$)	moveDependents(e_1, e')
genDeleteCons($Constraints_2$)	deleteEdge($\langle\langle e_1, N_1, N_1' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N_2, N' \rangle\rangle]$)
moveDependents(e_2, e')	genDeleteCons($Constraints_2'$)
deleteEdge($\langle\langle e_2, N_{1/2}, N_2' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N_{1/2}, N' \rangle\rangle]$)	moveDependents(e_2, e')
	deleteEdge($\langle\langle e_2, N_2, N_2' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N_2, N' \rangle\rangle]$)

$\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle = \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle$

$\langle\!\langle N_1 \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle N_2 \rangle\!\rangle$

$\texttt{createdNodal}(\langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle N \rangle\!\rangle)$

$\langle\!\langle N_1' \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle N_2' \rangle\!\rangle$

$\texttt{createdNodal}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle N' \rangle\!\rangle)$

$\texttt{uniqueName}(e_1, e_2, e', \overset{\mathrm{S}}{=})$

$\texttt{constraints}(\langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, Constraints_1)$

$\texttt{constraints}(\langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle, Constraints_2)$

$\texttt{commonCons}(Constraints_1, Constraints_2, Constraints)$

$\texttt{constraints}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, Constraints_1')$

$\texttt{constraints}(\langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle, Constraints_2')$

$\texttt{commonCons}(Constraints_1', Constraints_2', Constraints')$

$\mathsf{addEdge}(\langle\!\langle e', N, N' \rangle\!\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle])$

$\texttt{addConsList}(Constraints, \langle\!\langle N \rangle\!\rangle, \langle\!\langle e', N, N' \rangle\!\rangle)$

$\texttt{addConsList}(Constraints', \langle\!\langle N' \rangle\!\rangle, \langle\!\langle e', N, N' \rangle\!\rangle)$

$\texttt{genDeleteCons}(Constraints_1)$

$\texttt{genDeleteCons}(Constraints_1')$

$\texttt{moveDependents}(e_1, e')$

$\mathsf{deleteEdge}(\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e', N, N' \rangle\!\rangle])$

$\texttt{genDeleteCons}(Constraints_2)$

$\texttt{genDeleteCons}(Constraints_2')$

$\texttt{moveDependents}(e_2, e')$

$\mathsf{deleteEdge}(\langle\!\langle e_2, N_2, N_2' \rangle\!\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e', N, N' \rangle\!\rangle])$

The Optional Edge Removal rules are:

$\langle\langle e_2, N_{1/2}, N_2' \rangle\rangle \overset{S}{\subset} \langle\langle e_1, N_{1/2}, N_1' \rangle\rangle$

$\langle\langle N_2' \rangle\rangle \overset{S}{\subset} \langle\langle N_1' \rangle\rangle$

$\langle\langle e_1, N_{1/2}, N_1' \rangle\rangle \lhd \langle\langle N_1' \rangle\rangle$

$\langle\langle N_2' \rangle\rangle \rhd \langle\langle e_2, N_{1/2}, N_2' \rangle\rangle$

`constraints`$(\langle\langle N_2' \rangle\rangle, \langle\langle e_2, N_{1/2}, N_2' \rangle\rangle,$ *Constraints*$)$

`genDeleteCons`$(Constraints)$

`moveDependents`(e_2, e_1)

deleteEdge$(\langle\langle e_2, N_{1/2}, N_2' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e_1, N_{1/2}, N_1' \rangle\rangle; \{y\} \leftarrow \langle\langle N_2' \rangle\rangle)$

$\langle\langle e_2, N_2, N_2' \rangle\rangle \overset{S}{\subset} \langle\langle e_1, N_1, N_1' \rangle\rangle$

$\langle\langle N_2 \rangle\rangle \overset{S}{\subset} \langle\langle N_1 \rangle\rangle$

$\langle\langle N_2' \rangle\rangle \overset{S}{\subset} \langle\langle N_1' \rangle\rangle$

$\langle\langle e_1, N_1, N_1' \rangle\rangle \lhd \langle\langle N_1' \rangle\rangle$

$\langle\langle N_2' \rangle\rangle \rhd \langle\langle e_2, N_2, N_2' \rangle\rangle$

`constraints`$(\langle\langle N_2' \rangle\rangle, \langle\langle e_2, N_2, N_2' \rangle\rangle,$ *Constraints*$)$

`genDeleteCons`$(Constraints)$

`moveDependents`(e_2, e_1)

deleteEdge$(\langle\langle e_2, N_2, N_2' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e_1, N_1, N_1' \rangle\rangle; \{y\} \leftarrow \langle\langle N_2' \rangle\rangle)$

$\langle\langle e_2, N_2, N_2' \rangle\rangle \overset{S}{\subset} \langle\langle e_1, N_1, N_1' \rangle\rangle$

$\langle\langle N_2 \rangle\rangle \overset{S}{\cap} \langle\langle N_1 \rangle\rangle$

$\langle\langle N_2' \rangle\rangle \overset{S}{\subset} \langle\langle N_1' \rangle\rangle$

$\langle\langle e_1, N_1, N_1' \rangle\rangle \lhd \langle\langle N_1' \rangle\rangle$

$\langle\langle N_2' \rangle\rangle \rhd \langle\langle e_2, N_2, N_2' \rangle\rangle$

`constraints`$(\langle\langle N_2' \rangle\rangle, \langle\langle e_2, N_2, N_2' \rangle\rangle,$ *Constraints*$)$

`genDeleteCons`$(Constraints)$

`moveDependents`(e_2, e_1)

deleteEdge$(\langle\langle e_2, N_2, N_2' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e_1, N_1, N_1' \rangle\rangle; \{y\} \leftarrow \langle\langle N_2' \rangle\rangle)$

The Addition of Edge Intersection rules are:

$\langle\!\langle e_1, N_{1/2}, N'_{1/2} \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle e_2, N_{1/2}, N'_{1/2} \rangle\!\rangle$ $\texttt{uniqueName}(N_{1/2}, N_{1/2}, N', \overset{\mathrm{S}}{\cap})$ $\texttt{uniqueName}(N'_{1/2}, N'_{1/2}, N'', \overset{\mathrm{S}}{\cap})$ --- $\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid$ $\{x, y\} \leftarrow \langle\!\langle e_1, N_{1/2}, N'_{1/2} \rangle\!\rangle;$ $\{x, y\} \leftarrow \langle\!\langle e_2, N_{1/2}, N'_{1/2} \rangle\!\rangle)]$ $\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_{1/2} \rangle\!\rangle \rangle\!\rangle)$ $\mathsf{addNode}(\langle\!\langle N'' \rangle\!\rangle, [\{x\} \mid$ $\{y, x\} \leftarrow \langle\!\langle e_1, N_{1/2}, N'_{1/2} \rangle\!\rangle;$ $\{y, x\} \leftarrow \langle\!\langle e_2, N_{1/2}, N'_{1/2} \rangle\!\rangle)]$ $\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N'' \rangle\!\rangle, \langle\!\langle N'_{1/2} \rangle\!\rangle \rangle\!\rangle)$	$\langle\!\langle e_1, N_{1/2}, N'_1 \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle e_2, N_{1/2}, N'_2 \rangle\!\rangle$ $\langle\!\langle N'_2 \rangle\!\rangle \overset{\mathrm{S}}{\subset} \langle\!\langle N'_1 \rangle\!\rangle$ $\texttt{uniqueName}(N_{1/2}, N_{1/2}, N', \overset{\mathrm{S}}{\cap})$ $\texttt{uniqueName}(N'_1, N'_2, N'', \overset{\mathrm{S}}{\cap})$ --- $\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid$ $\{x, y\} \leftarrow \langle\!\langle e_1, N_{1/2}, N'_1 \rangle\!\rangle;$ $\{x, y\} \leftarrow \langle\!\langle e_2, N_{1/2}, N'_2 \rangle\!\rangle)]$ $\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_{1/2} \rangle\!\rangle \rangle\!\rangle)$ $\mathsf{addNode}(\langle\!\langle N'' \rangle\!\rangle, [\{x\} \mid$ $\{y, x\} \leftarrow \langle\!\langle e_1, N_{1/2}, N'_1 \rangle\!\rangle;$ $\{y, x\} \leftarrow \langle\!\langle e_2, N_{1/2}, N'_2 \rangle\!\rangle)]$ $\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N'' \rangle\!\rangle, \langle\!\langle N'_2 \rangle\!\rangle \rangle\!\rangle)$
$\langle\!\langle e_1, N_1, N'_1 \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle e_2, N_2, N'_2 \rangle\!\rangle$ $\langle\!\langle N_2 \rangle\!\rangle \overset{\mathrm{S}}{\subset} \langle\!\langle N_1 \rangle\!\rangle$ $\langle\!\langle N'_2 \rangle\!\rangle \overset{\mathrm{S}}{\subset} \langle\!\langle N'_1 \rangle\!\rangle$ $\texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\cap})$ $\texttt{uniqueName}(N'_1, N'_2, N'', \overset{\mathrm{S}}{\cap})$ --- $\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid$ $\{x, y\} \leftarrow \langle\!\langle e_1, N_1, N'_1 \rangle\!\rangle;$ $\{x, y\} \leftarrow \langle\!\langle e_2, N_2, N'_2 \rangle\!\rangle)]$ $\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle \rangle\!\rangle)$ $\mathsf{addNode}(\langle\!\langle N'' \rangle\!\rangle, [\{x\} \mid$ $\{y, x\} \leftarrow \langle\!\langle e_1, N_1, N'_1 \rangle\!\rangle;$ $\{y, x\} \leftarrow \langle\!\langle e_2, N_2, N'_2 \rangle\!\rangle)]$ $\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N'' \rangle\!\rangle, \langle\!\langle N'_2 \rangle\!\rangle \rangle\!\rangle)$	$\langle\!\langle e_1, N_1, N'_1 \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle e_2, N_2, N'_2 \rangle\!\rangle$ $\langle\!\langle N_1 \rangle\!\rangle \overset{\mathrm{S}}{\subset} \langle\!\langle N_2 \rangle\!\rangle$ $\langle\!\langle N'_2 \rangle\!\rangle \overset{\mathrm{S}}{\subset} \langle\!\langle N'_1 \rangle\!\rangle$ $\texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\cap})$ $\texttt{uniqueName}(N'_1, N'_2, N'', \overset{\mathrm{S}}{\cap})$ --- $\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid$ $\{x, y\} \leftarrow \langle\!\langle e_1, N_1, N'_1 \rangle\!\rangle;$ $\{x, y\} \leftarrow \langle\!\langle e_2, N_2, N'_2 \rangle\!\rangle)]$ $\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle \rangle\!\rangle)$ $\mathsf{addNode}(\langle\!\langle N'' \rangle\!\rangle, [\{x\} \mid$ $\{y, x\} \leftarrow \langle\!\langle e_1, N_1, N'_1 \rangle\!\rangle;$ $\{y, x\} \leftarrow \langle\!\langle e_2, N_2, N'_2 \rangle\!\rangle)]$ $\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N'' \rangle\!\rangle, \langle\!\langle N'_2 \rangle\!\rangle \rangle\!\rangle)$

$\langle\!\langle e_1, N_{1/2}, N_1' \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle e_2, N_{1/2}, N_2' \rangle\!\rangle$

$\langle\!\langle N_1' \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle N_2' \rangle\!\rangle$

$\texttt{createdNodal}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle N \rangle\!\rangle)$

$\texttt{uniqueName}(N_{1/2}, N_{1/2}, N', \overset{\mathrm{S}}{\cap})$

$\texttt{uniqueName}(N_1', N_2', N'', \overset{\mathrm{S}}{\cap})$

$\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e_1, N_{1/2}, N_1' \rangle\!\rangle;$

$\{x, y\} \leftarrow \langle\!\langle e_2, N_{1/2}, N_2' \rangle\!\rangle)]$

$\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_{1/2} \rangle\!\rangle \rangle\!\rangle)$

$\mathsf{addNode}(\langle\!\langle N'' \rangle\!\rangle, [\{x\} \mid$

$\{y, x\} \leftarrow \langle\!\langle e_1, N_{1/2}, N_1' \rangle\!\rangle;$

$\{y, x\} \leftarrow \langle\!\langle e_2, N_{1/2}, N_2' \rangle\!\rangle)]$

$\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N'' \rangle\!\rangle, \langle\!\langle N \rangle\!\rangle \rangle\!\rangle)$

$\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle$

$\langle\!\langle N_2 \rangle\!\rangle \subset \langle\!\langle N_1 \rangle\!\rangle$

$\langle\!\langle N_1' \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle N_2' \rangle\!\rangle$

$\texttt{createdNodal}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle N \rangle\!\rangle)$

$\texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\cap})$

$\texttt{uniqueName}(N_1', N_2', N'', \overset{\mathrm{S}}{\cap})$

$\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle;$

$\{x, y\} \leftarrow \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle)]$

$\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle \rangle\!\rangle)$

$\mathsf{addNode}(\langle\!\langle N'' \rangle\!\rangle, [\{x\} \mid$

$\{y, x\} \leftarrow \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle;$

$\{y, x\} \leftarrow \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle)]$

$\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N'' \rangle\!\rangle, \langle\!\langle N \rangle\!\rangle \rangle\!\rangle)$

$\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle$

$\langle\!\langle N_1 \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle N_2 \rangle\!\rangle$

$\texttt{createdNodal}(\langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle N_{12} \rangle\!\rangle)$

$\langle\!\langle N_1' \rangle\!\rangle \overset{\mathrm{S}}{\cap} \langle\!\langle N_2' \rangle\!\rangle$

$\texttt{createdNodal}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle N_{12}' \rangle\!\rangle)$

$\texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\cap})$

$\texttt{uniqueName}(N_1', N_2', N'', \overset{\mathrm{S}}{\cap})$

$\mathsf{addNode}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid$

$\{x, y\} \leftarrow \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle;$

$\{x, y\} \leftarrow \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle)]$

$\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_{12} \rangle\!\rangle \rangle\!\rangle)$

$\mathsf{addNode}(\langle\!\langle N'' \rangle\!\rangle, [\{x\} \mid$

$\{y, x\} \leftarrow \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle;$

$\{y, x\} \leftarrow \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle)]$

$\mathsf{addConstraint}(\langle\!\langle \subseteq, \langle\!\langle N'' \rangle\!\rangle, \langle\!\langle N_{12}' \rangle\!\rangle \rangle\!\rangle)$

The Generalization of Edges rules are:

$\langle\!\langle e_1, N_{1/2}, N_1' \rangle\!\rangle \overset{\mathrm{S}}{\sqcap} \langle\!\langle e_2, N_{1/2}, N_2' \rangle\!\rangle$

$\langle\!\langle N_1' \rangle\!\rangle \overset{\mathrm{S}}{\sqcap} \langle\!\langle N_2' \rangle\!\rangle$

$\texttt{createdNodal}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle N' \rangle\!\rangle)$

$\texttt{uniqueName}(e_1, e_2, e', \overset{\mathrm{S}}{\sqcap})$

$\texttt{constraints}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle e_1, N_{1/2}, N_1' \rangle\!\rangle, Constraints_1')$

$\texttt{constraints}(\langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle e_2, N_{1/2}, N_2' \rangle\!\rangle, Constraints_2')$

$\texttt{commonCons}(Constraints_1', Constraints_2', Constraints')$

$\texttt{constraints}(\langle\!\langle N_{1/2} \rangle\!\rangle, \langle\!\langle e_1, N_{1/2}, N_1' \rangle\!\rangle, Constraints_1)$

$\texttt{constraints}(\langle\!\langle N_{1/2} \rangle\!\rangle, \langle\!\langle e_2, N_{1/2}, N_2' \rangle\!\rangle, Constraints_2)$

$\texttt{commonCons}(Constraints_1, Constraints_2, Constraints)$

$\texttt{addEdge}(\langle\!\langle e', N_{1/2}, N' \rangle\!\rangle, \langle\!\langle e_1, N_{1/2}, N_1' \rangle\!\rangle \texttt{++} \langle\!\langle e_2, N_{1/2}, N_2' \rangle\!\rangle)$

$\texttt{addConsList}(Constraints, \langle\!\langle N_{1/2} \rangle\!\rangle, \langle\!\langle e', N_{1/2}, N' \rangle\!\rangle)$

$\texttt{addConsList}(Constraints', \langle\!\langle N' \rangle\!\rangle, \langle\!\langle e', N_{1/2}, N' \rangle\!\rangle)$

$\texttt{genDeleteCons}(Constraints_1)$

$\texttt{moveDependents}(e_1, e')$

$\texttt{deleteEdge}(\langle\!\langle e_1, N_{1/2}, N_1' \rangle\!\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\!\langle e', N_{1/2}, N' \rangle\!\rangle; \{y\} \leftarrow \langle\!\langle N_1' \rangle\!\rangle])$

$\texttt{genDeleteCons}(Constraints_2)$

$\texttt{moveDependents}(e_2, e')$

$\langle\!\langle e_1, N_1, N_1' \rangle\!\rangle \overset{\mathrm{S}}{\sqcap} \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle$

$\langle\!\langle N_2 \rangle\!\rangle \overset{\mathrm{S}}{\sqsubset} \langle\!\langle N_1 \rangle\!\rangle$

$\langle\!\langle N_1' \rangle\!\rangle \overset{\mathrm{S}}{\sqcap} \langle\!\langle N_2' \rangle\!\rangle$

$\texttt{createdNodal}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle N_2' \rangle\!\rangle, \langle\!\langle N' \rangle\!\rangle)$

$\texttt{uniqueName}(e_1, e_2, e', \overset{\mathrm{S}}{\sqcap})$

$\texttt{constraints}(\langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, Constraints_1)$

$\texttt{constraints}(\langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle e_2, N_2, N_2' \rangle\!\rangle, Constraints_2)$

$\texttt{commonCons}(Constraints_1, Constraints_2, Constraints)$

$\texttt{constraints}(\langle\!\langle N_1' \rangle\!\rangle, \langle\!\langle e_1, N_1, N_1' \rangle\!\rangle, Constraints_1')$

$\texttt{constraints}(\langle\langle N_2' \rangle\rangle, \langle\langle e_2, N_2, N_2' \rangle\rangle, Constraints_2')$

$\texttt{commonCons}(Constraints_1', Constraints_2', Constraints')$

$\mathsf{addEdge}(\langle\langle e', N_1, N' \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle \texttt{++} \langle\langle e_2, N_2, N_2' \rangle\rangle)$

$\texttt{addConsList}(Constraints, \langle\langle N_1 \rangle\rangle, \langle\langle e', N_1, N' \rangle\rangle)$

$\texttt{addConsList}(Constraints', \langle\langle N' \rangle\rangle, \langle\langle e', N_1, N' \rangle\rangle)$

$\texttt{genDeleteCons}(Constraints_1)$

$\texttt{moveDependents}(e_1, e')$

$\mathsf{deleteEdge}(\langle\langle e_1, N_1, N_1' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N_1, N' \rangle\rangle; \{y\} \leftarrow \langle\langle N_1' \rangle\rangle])$

$\texttt{genDeleteCons}(Constraints_2)$

$\texttt{moveDependents}(e_2, e')$

$\mathsf{deleteEdge}(\langle\langle e_2, N_2, N_2' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N_1, N' \rangle\rangle; \{y\} \leftarrow \langle\langle N_2' \rangle\rangle])$

$\langle\langle e_1, N_1, N_1' \rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle e_2, N_2, N_2' \rangle\rangle$

$\langle\langle N_1 \rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle N_2 \rangle\rangle$

$\texttt{createdNodal}(\langle\langle N_1 \rangle\rangle, \langle\langle N_2 \rangle\rangle, \langle\langle N_{12} \rangle\rangle)$

$\langle\langle N_1' \rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle N_2' \rangle\rangle$

$\texttt{createdNodal}(\langle\langle N_1' \rangle\rangle, \langle\langle N_2' \rangle\rangle, \langle\langle N'' \rangle\rangle)$

$\texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\sqcap})$

$\texttt{uniqueName}(e_1, e_2, e', \overset{\mathrm{S}}{\sqcap})$

$\texttt{constraints}(\langle\langle N_1 \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle, Constraints_1)$

$\texttt{constraints}(\langle\langle N_2 \rangle\rangle, \langle\langle e_2, N_2, N_2' \rangle\rangle, Constraints_2)$

$\texttt{commonCons}(Constraints_1, Constraints_2, Constraints)$

$\texttt{constraints}(\langle\langle N_1' \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle, Constraints_1')$

$\texttt{constraints}(\langle\langle N_2' \rangle\rangle, \langle\langle e_2, N_2, N_2' \rangle\rangle, Constraints_2')$

$\texttt{commonCons}(Constraints_1', Constraints_2', Constraints')$

$\mathsf{addNode}(\langle\langle N' \rangle\rangle, \langle\langle N_1 \rangle\rangle \texttt{++} \langle\langle N_2 \rangle\rangle)$

$\mathsf{addEdge}(\langle\langle e', N', N'' \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle \texttt{++} \langle\langle e_2, N_2, N_2' \rangle\rangle)$

$\texttt{addConsList}(Constraints, \langle\langle N' \rangle\rangle, \langle\langle e', N', N'' \rangle\rangle)$

$\texttt{addConsList}(Constraints', \langle\langle N'' \rangle\rangle, \langle\langle e', N', N'' \rangle\rangle)$

genDeleteCons($Constraints_1$)

moveDependents(e_1,e')

deleteEdge($\langle\langle e_1, N_1, N_1' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N', N'' \rangle\rangle; \{y\} \leftarrow \langle\langle N_1' \rangle\rangle]$)

genDeleteCons($Constraints_2$)

moveDependents(e_2,e')

deleteEdge($\langle\langle e_2, N_2, N_2' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N', N'' \rangle\rangle; \{y\} \leftarrow \langle\langle N_2' \rangle\rangle]$)

$\langle\langle e_1, N_1, N_1' \rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle e_2, N_2, N_2' \rangle\rangle$

$\langle\langle N_1 \rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle N_2 \rangle\rangle$

createdNodal($\langle\langle N_1 \rangle\rangle, \langle\langle N_2 \rangle\rangle, \langle\langle N' \rangle\rangle$)

$\langle\langle N_1' \rangle\rangle \overset{\mathrm{S}}{\sqcap} \langle\langle N_2' \rangle\rangle$

createdNodal($\langle\langle N_1' \rangle\rangle, \langle\langle N_2' \rangle\rangle, \langle\langle N'' \rangle\rangle$)

uniqueName($e_1, e_2, e', \overset{\mathrm{S}}{\sqcap}$)

constraints($\langle\langle N_1 \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle, Constraints_1$)

constraints($\langle\langle N_2 \rangle\rangle, \langle\langle e_2, N_2, N_2' \rangle\rangle, Constraints_2$)

commonCons($Constraints_1, Constraints_2, Constraints$)

constraints($\langle\langle N_1' \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle, Constraints_1'$)

constraints($\langle\langle N_2' \rangle\rangle, \langle\langle e_2, N_2, N_2' \rangle\rangle, Constraints_2'$)

commonCons($Constraints_1', Constraints_2', Constraints'$)

---

addEdge($\langle\langle e', N', N'' \rangle\rangle, \langle\langle e_1, N_1, N_1' \rangle\rangle$++$\langle\langle e_2, N_2, N_2' \rangle\rangle$)

addConsList($Constraints, \langle\langle N' \rangle\rangle, \langle\langle e', N', N'' \rangle\rangle$)

addConsList($Constraints', \langle\langle N'' \rangle\rangle, \langle\langle e', N', N'' \rangle\rangle$)

genDeleteCons($Constraints_1$)

moveDependents(e_1,e')

deleteEdge($\langle\langle e_1, N_1, N_1' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N', N'' \rangle\rangle; \{x\} \leftarrow \langle\langle N_1 \rangle\rangle]$)

genDeleteCons($Constraints_2$)

moveDependents(e_2,e')

deleteEdge($\langle\langle e_2, N_2, N_2' \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle e', N', N'' \rangle\rangle; \{x\} \leftarrow \langle\langle N_2 \rangle\rangle]$)

Appendix B

Generic Merging Rules

In this appendix we list all generic merging rules for nodals and link-nodals. The rules about links are identical to the low-level rules on edges.

B.1 Naming Conforming Rules

$\langle\langle\langle N_1\rangle\rangle, \stackrel{\mathrm{S}}{=}, \langle\langle N_2\rangle\rangle\rangle$	$\neg\ \langle\langle\langle N_1\rangle\rangle, \stackrel{\mathrm{S}}{=}, \langle\langle N_2\rangle\rangle\rangle$
$\neg$ identicalNames(N_1, N_2)	identicalNames(N_1, N_2)
commonName(N_1, N_2, N')	distinctNames(N_1, N_2, N_1', N_2')
renameNodal$_{gen}(\langle\langle N_1\rangle\rangle, \langle\langle N'\rangle\rangle)$	renameNodal$_{gen}(\langle\langle N_1\rangle\rangle, \langle\langle N_1'\rangle\rangle)$
renameNodal$_{gen}(\langle\langle N_2\rangle\rangle, \langle\langle N'\rangle\rangle)$	renameNodal$_{gen}(\langle\langle N_2\rangle\rangle, \langle\langle N_2'\rangle\rangle)$

$LN_1 \stackrel{\mathrm{S}}{=} LN_2$	$\neg\ LN_1 \stackrel{\mathrm{S}}{=} LN_2$
$\neg$ identicalNames(LN_1, LN_2)	identicalNames(LN_1, LN_2)
commonName(LN_1, LN_2, LN')	distinctNames$(LN_1, LN_2, LN_1', LN_2')$
renameLN$_{gen}(LN_1, LN')$	renameLN$_{gen}(LN_1, LN_1')$
renameLN$_{gen}(LN_2, LN')$	renameLN$_{gen}(LN_2, LN_2')$

B.2 Unioning

$\langle \langle\!\langle N_2 \rangle\!\rangle, \overset{\mathrm{S}}{\subset}, \langle\!\langle N_1 \rangle\!\rangle \rangle$

$\texttt{addConstraint}_{gen}(\langle\!\langle \subseteq, \langle\!\langle N_2 \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle \rangle\!\rangle)$

$\langle \langle\!\langle N_1 \rangle\!\rangle, \overset{\mathrm{S}}{\cap}, \langle\!\langle N_2 \rangle\!\rangle \rangle$

$\texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\cap})$

$\texttt{addNodal}_{gen}(\langle\!\langle N' \rangle\!\rangle, [\{x\} \mid \{x\} \leftarrow \langle\!\langle N_1 \rangle\!\rangle;$
$\{x\} \leftarrow \langle\!\langle N_2 \rangle\!\rangle])$
$\texttt{addConstraint}_{gen}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle \rangle\!\rangle)$
$\texttt{addConstraint}_{gen}(\langle\!\langle \subseteq, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle \rangle\!\rangle)$

$\langle \langle\!\langle N_1 \rangle\!\rangle, \overset{\mathrm{S}}{\not\cap}, \langle\!\langle N_2 \rangle\!\rangle \rangle$

$\texttt{uniqueName}(N_1, N_2, N', \overset{\mathrm{S}}{\not\cap})$

$\texttt{addNodal}_{gen}(\langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle \texttt{ ++ } \langle\!\langle N_2 \rangle\!\rangle)$
$\texttt{addConstraint}_{gen}($
$\langle\!\langle \cup, \langle\!\langle N' \rangle\!\rangle, \langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle \rangle\!\rangle)$
$\texttt{addConstraint}_{gen}(\langle\!\langle \not\cap, \langle\!\langle N_1 \rangle\!\rangle, \langle\!\langle N_2 \rangle\!\rangle \rangle\!\rangle)$

B.3 Restructuring

The Redundant Link-Nodal Removal rule is:

$\langle\!\langle X_1, N_1 \rangle\!\rangle \overset{\mathrm{S}}{=} \langle\!\langle X_2, N_2 \rangle\!\rangle$

$\langle\!\langle X_2 \rangle\!\rangle \overset{\mathrm{S}}{\subset} \langle\!\langle X_1 \rangle\!\rangle$

$\texttt{constraints}(\langle\!\langle X_1 \rangle\!\rangle, \langle\!\langle X_1, N_1 \rangle\!\rangle, \mathit{Constraints})$

$\texttt{genDeleteCons}(\mathit{Constraints})$

$\texttt{deleteLN}_{gen}(\langle\!\langle X_1, N_1 \rangle\!\rangle, [\{x, y\} \mid$
$\{x, y\} \leftarrow \langle\!\langle X_2, N_2 \rangle\!\rangle])$

The Specialization of Link-Nodals rule is:

$\langle\langle X_1, N_1\rangle\rangle \overset{\mathrm{S}}{=} \langle\langle X_2, N_2\rangle\rangle$

$\langle\langle X_1\rangle\rangle \overset{\mathrm{S}}{\cap} \langle\langle X_2\rangle\rangle$

`createdNodal`$(\langle\langle X_1\rangle\rangle, \langle\langle X_2\rangle\rangle, \langle\langle X'\rangle\rangle)$

`uniqueName`$(N_1, N_2, N', \overset{\mathrm{S}}{=})$

`constraints`$(\langle\langle X_1\rangle\rangle, \langle\langle X_1, N_1\rangle\rangle, Constraints_1)$

`constraints`$(\langle\langle X_2\rangle\rangle, \langle\langle X_2, N_2\rangle\rangle, Constraints_2)$

`commonCons`$(Constraints_1, Constraints_2, Constraints)$

`addLN`$_{gen}(\langle\langle X', N'\rangle\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\langle X_1, N_1\rangle\rangle])$

`addConsList`$(Constraints, \langle\langle X'\rangle\rangle, \langle\langle X', N'\rangle\rangle)$

`genDeleteCons`$(Constraints_1)$

`deleteLN`$_{gen}(\langle\langle X_1, N_1\rangle\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\langle X', N'\rangle\rangle])$

`genDeleteCons`$(Constraints_2)$

`deleteLN`$_{gen}(\langle\langle X_2, N_2\rangle\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\langle X', N'\rangle\rangle])$

The Optional Link-Nodal Removal rule is:

$\langle\langle X_2, N_2\rangle\rangle \overset{\mathrm{S}}{\subset} \langle\langle X_1, N_1\rangle\rangle$

$\langle\langle X_2\rangle\rangle \overset{\mathrm{S}}{\subset} \langle\langle X_1\rangle\rangle$

$\langle\langle X_1\rangle\rangle \lhd \langle\langle X_1, N_1\rangle\rangle$

$\langle\langle X_1\rangle\rangle \rhd \langle\langle X_1, N_1\rangle\rangle$

$\langle\langle X_2\rangle\rangle \rhd \langle\langle X_2, N_2\rangle\rangle$

`deleteLN`$_{gen}(\langle\langle X_2, N_2\rangle\rangle, [\{x, y\} \mid$

$\{x, y\} \leftarrow \langle\langle X_1, N_1\rangle\rangle; \{x\} \leftarrow \langle\langle X_2\rangle\rangle])$

The Addition of Link-Nodal Intersection rules are:

$\langle\langle X_{1/2}, N_1\rangle\rangle \overset{S}{\cap} \langle\langle X_{1/2}, N_2\rangle\rangle$

$\texttt{uniqueName}(X_{1/2}, X_{1/2}, X', \overset{S}{\cap})$

$\texttt{addNodal}_{gen}(\langle\langle X'\rangle\rangle, [\{x\} \mid$

$\{x, y\} \leftarrow \langle\langle X_{1/2}, N_1\rangle\rangle;$

$\{x, y\} \leftarrow \langle\langle X_{1/2}, N_2\rangle\rangle)]$

$\texttt{addConstraint}_{gen}($

$\langle\langle \subseteq, \langle\langle X'\rangle\rangle, \langle\langle X_{1/2}\rangle\rangle\rangle\rangle)$

$\langle\langle X_1, N_1\rangle\rangle \overset{S}{\cap} \langle\langle X_2, N_2\rangle\rangle$

$\langle\langle X_2\rangle\rangle \overset{S}{\subset} \langle\langle X_1\rangle\rangle$

$\texttt{uniqueName}(X_1, X_2, X', \overset{S}{\cap})$

$\texttt{addNodal}_{gen}(\langle\langle X'\rangle\rangle, [\{x\} \mid$

$\{x, y\} \leftarrow \langle\langle X_1, N_1\rangle\rangle;$

$\{x, y\} \leftarrow \langle\langle X_2, N_2\rangle\rangle)]$

$\texttt{addConstraint}_{gen}(\langle\langle \subseteq, \langle\langle X'\rangle\rangle, \langle\langle X_2\rangle\rangle\rangle\rangle)$

$\langle\langle X_1, N_1\rangle\rangle \overset{S}{\cap} \langle\langle X_2, N_2\rangle\rangle$

$\langle\langle X_1\rangle\rangle \overset{S}{\cap} \langle\langle X_2\rangle\rangle$

$\texttt{createdNodal}(\langle\langle X_1\rangle\rangle, \langle\langle X_2\rangle\rangle, \langle\langle X'\rangle\rangle)$

$\texttt{uniqueName}(X', X', X'', \overset{S}{\cap})$

$\texttt{addNodal}_{gen}(\langle\langle X''\rangle\rangle, [\{x\} \mid$

$\{x, y\} \leftarrow \langle\langle X_1, N_1\rangle\rangle;$

$\{x, y\} \leftarrow \langle\langle X_2, N_2\rangle\rangle)]$

$\texttt{addConstraint}_{gen}(\langle\langle \subseteq, \langle\langle X''\rangle\rangle, \langle\langle X'\rangle\rangle\rangle\rangle)$

The Link-Nodal Generalization rule is:

$\langle\langle X_1, N_1\rangle\rangle \overset{S}{\not\cap} \langle\langle X_2, N_2\rangle\rangle$

$\langle\langle X_1\rangle\rangle \overset{S}{\not\cap} \langle\langle X_2\rangle\rangle$

$\texttt{createdNodal}(\langle\langle X_1\rangle\rangle, \langle\langle X_2\rangle\rangle, \langle\langle X'\rangle\rangle)$

$\texttt{uniqueName}(N_1, N_2, N', \overset{S}{\not\cap})$

$\texttt{constraints}(\langle\langle X_1\rangle\rangle, \langle\langle X_1, N_1\rangle\rangle, Constraints_1)$

$\texttt{constraints}(\langle\langle X_2\rangle\rangle, \langle\langle X_2, N_2\rangle\rangle, Constraints_2)$

$\texttt{commonCons}(Constraints_1, Constraints_2, Constraints)$

$\texttt{addLN}_{gen}(\langle\langle X', N'\rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle X_1, N_1\rangle\rangle]\texttt{++}[\{x, y\} \mid \{x, y\} \leftarrow \langle\langle X_2, N_2\rangle\rangle])$

$\texttt{addConsList}(Constraints, \langle\langle X'\rangle\rangle, \langle\langle X', N'\rangle\rangle)$

$\texttt{deleteLN}_{gen}(\langle\langle X_1, N_1\rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle X', N'\rangle\rangle; \{x\} \leftarrow \langle\langle X_1\rangle\rangle])$

$\texttt{deleteLN}_{gen}(\langle\langle X_2, N_2\rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle X', N'\rangle\rangle; \{x\} \leftarrow \langle\langle X_2\rangle\rangle])$

Bibliography

[1] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, and Aristides Gionis. Automated ranking of database query results. In *CIDR*, pages 888–899, 2003.

[2] Ken Barker amon Lawrence. Integrating relational database schemas using a standardized dictionary. In *ACM Symposium on Applied Computing (SAC)*, pages 225–230, 2001.

[3] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.

[4] Marcelo Arenas and Leonid Libkin. XML data exchange: consistency and query answering. In *PODS*, pages 13–24, 2005.

[5] David Aumueller, Hong Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *SIGMOD*, pages 906–908, 2005.

[6] Michelle Q. Wang Baldonado, Kevin Chen-Chuan Chang, Luis Gravano, and Andreas Paepcke. The Stanford digital library metadata architecture. *Int. J. on Digital Libraries*, 1(2):108–121, 1997.

[7] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, 10(6):650–664, 1984.

[8] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[9] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Beneventano. Semantic integration of heterogeneous information sources. *Data Knowl. Eng.*, 36(3):215–249, 2001.

[10] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *CAiSE*, pages 452–466, 2002.

[11] Philip A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, pages 209–222, 2003.

[12] J. Biskup and B. Convent. A formal view integration method. In *SIGMOD*, pages 398–407, 1986.

[13] Dina Bitton and David J. DeWitt. Duplicate record elimination in large data files. *ACM Trans. Database Syst.*, 8(2):255–265, 1983.

[14] Shawn Bowers and Lois M. L. Delcambre. The uni-level description: A uniform framework for representing information in multiple data models. In *ER*, pages 45–58, 2003.

[15] M. Boyd, S. Kittivoravitkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. AutoMed: A BAV data integration system for heterogeneous data sources. In *CAiSE*, pages 82–97, 2004.

[16] Michael Boyd and Peter McBrien. Comparing and transforming between data models via an intermediate hypergraph data model. *J. Data Semantics IV*, pages 69–109, 2005.

[17] Tim Bray and C. M. Sperberg-McQueen. Extensible markup language (XML), 1996. http://www.w3.org/TR/WD-xml-961114.html.

[18] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *EDBT*, volume 580 of *LNCS*, pages 152–167, 1992.

[19] Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.

[20] Arnon Rosenthal C. Clifton, E. Housman. Experience with a combined approach to attribute-matching across heterogeneous databases. In *DS-7*, volume 124 of *IFIP Conference Proceedings*, pages 429–451, 1998.

[21] M.A. Casanova and V.M.P. Vidal. Towards a sound view integration methodology. In *SIGACT–SIGMOD*, pages 36–47, 1983.

[22] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and systems.* McGraw-Hill, 1994.

[23] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.

[24] P.P. Chen. The Entity-Relationship model — toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[25] Laura Chiticariu and Wang Chiew Tan. Debugging schema mappings with routes. In *VLDB*, pages 79–90, 2006.

[26] Kajal T. Claypool and Elke A. Rundensteiner. Sangam: A framework for modeling heterogeneous database transformations. In *ICEIS (1)*, pages 219–224, 2003.

[27] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[28] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, 8th edition edition, 2004.

[29] C.J. Date, H. Darwen, and N.A. Lorentzos. *Temporal Data and the Relational Model*. Morgan Kaufmann, 2003.

[30] Thomas Devogele, Christine Parent, and Stefano Spaccapietra. On spatial database integration. *International Journal of Geographical Information Science*, 12(4):335–352, 1998.

[31] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Y. Halevy, and Pedro Domingos. iMAP: Discovering complex mappings between database schemas. In *SIGMOD*, pages 383–394, 2004.

[32] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems*, pages 221–237, 2002.

[33] H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[34] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map ontologies on the Semantic Web. In *WWW*, pages 662–673, 2002.

[35] AnHai Doan, Pedro Domingos, and Alon Y. Levy. Learning source description for data integration. In *WebDB*, pages 81–86, 2000.

[36] Carmel Domshlak, Avigdor Gal, and Haggai Roitman. Rank aggregation for automatic schema matching. *IEEE Trans. Knowl. Data Eng.*, 19(4):538–553, 2007.

[37] Xin Luna Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. In *VLDB*, pages 687–698, 2007.

[38] David W. Embley, David Jackman, and Li Xu. Multifaceted exploitation of metadata for attribute match discovery in information integration. In *Workshop on Information Integration on the Web*, pages 110–117, 2001.

[39] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[40] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. In *PODS*, pages 90–101, 2003.

[41] P. Fankhauser, M. Kracker, and E. J. Neuhold. Semantics vs Structural Resemblance of Classes. *SIGMOD Record*, 20(4):59–63, 1991.

[42] Christiane Fellbaum, editor. *WordNet - An Electronic Lexical Database*. MIT Press, 1998.

[43] C. Francalanci and B. Pernici. View integration: A survey of current developments. Technical Report 93-053, Dipartimento di Elettronica e Informazione, P.zza Leonardo da Vinci 32, 20133 Milano, Italy, 1993.

[44] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *16th National Conference on Artificial Intelligence*, pages 67–73, 1999.

[45] Avigdor Gal. Why is schema matching tough and what can we do about it? *SIGMOD Record*, 35(4):2–5, 2006.

[46] Avigdor Gal, Ateret Anaby-Tavor, Alberto Trombetta, and Danilo Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 14(1):50–67, 2005.

[47] F. Hakimpour and A. Geppert. Global schema generation using formal ontologies. In *ER*, volume 2503 of *LNCS*, pages 307–321, 2002.

[48] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *WWW*, pages 556–567, 2003.

[49] Alon Y. Halevy, Michael J. Franklin, and David Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.

[50] Patrick A. V. Hall, J. Owlett, and Stephen Todd. Relations and entities. In *IFIP Working Conference on Modelling in Data Base Management Systems*, pages 201–220, 1976.

[51] Stephen Hayne and Sudha Ram. Multi-user view integration system (MUVIS): An expert system for view integration. In *ICDE*, pages 402–409, 1990.

[52] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[53] Lieming Huang, Matthias Hemmje, and Erich J. Neuhold. Admire: an adaptive data model for meta search engines. *Computer Networks*, 33(1-6):431–448, 2000.

[54] R. Hull. Managing sematic heterogeneity in databases: A theoretical perspective. In *PODS*, pages 51–61, 1997.

[55] V. Kashyap and A. Sheth. Semantic and schematic similarities between database objects: a context-based approach. *VLDB Journal*, 5(4):276–304, 1996.

[56] David Kensche, Christoph Quix, Mohamed Amine Chatti, and Matthias Jarke. Gerome: A generic role based metamodel for model management. *Journal of Data Semantics*, 8:82–117, 2007.

[57] Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, pages 61–75, 2005.

[58] Laks V. S. Lakshmanan, Nicola Leone, Robert B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

[59] J.A. Larson, S.B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, April 1989.

[60] Maurizio Lenzerini. Data integration: A theoretical perspective. In *ACM SIGACT-SIGMOD-SIGART*, pages 233–246, 2002.

[61] Alon Y. Levy. Logic-based techniques in data integration. *Logic-based artificial intelligence*, pages 575–595, 2000.

[62] Wen-Syan Li and Chris Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33:49–84, 2000.

[63] Dekang Lin. An information-theoretic definition of similarity. In *ICML*, pages 296–304, 1998.

[64] M. Castellanos M. Garcia-Solaco and F. Saltor. Discovering Interdatabase Resemblance of Classes for Interoperable Databases. In *RIDE-IMS*, pages 26–33, 1993.

[65] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with Cupid. In *VLDB*, pages 49–58, 2001.

[66] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[67] Jayant Madhavan and Alon Y. Halevy. Composing mappings among data sources. In *VLDB*, pages 572–583, 2003.

[68] Matteo Magnani, Nikos Rizopoulos, Peter McBrien, and Danilo Montesi. Schema integration based on uncertain semantic mappings. In *ER*, pages 31–46, 2005.

[69] Lu Mao, Khalid Belhajjame, Norman W. Paton, and Alvaro A. A. Fernandes. Defining and using schematic correspondences for automatically generating schema mappings. In *CAiSE*, pages 79–93, 2009.

[70] P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *CAiSE*, volume 1626 of *LNCS*, pages 333–348, 1999.

[71] P.J. McBrien and A. Poulovassilis. A semantic approach to integrating XML and structured data sources. In *CAiSE*, volume 2068 of *LNCS*, pages 330–345, 2001.

[72] P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *ICDE*, pages 227–238, 2003.

[73] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.

[74] Sergey Melnik, Philip A. Bernstein, Alon Y. Halevy, and Erhard Rahm. Supporting executable mappings in model management. In *SIGMOD*, pages 167–178, 2005.

[75] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: a programming platform for generic model management. In *SIGMOD*, pages 193–204, 2003.

[76] Renée J. Miller, Laura M. Haas, and Mauricio A. Hernández. Schema mapping as query discovery. In *VLDB*, pages 77–88, 2000.

[77] T. Mitchel. *Machine Learning*. McGraw-Hill, 1997.

[78] Isi Mitrani. *Probabilistic modelling*. Cambridge University Press, 1998.

[79] A. Motro and P. Buneman. Constructing superviews. In *SIGMOD*, pages 54–64, 1981.

[80] Shamkant B. Navathe, Ramez Elmasri, and James A. Larson. Integrating user views in database design. *IEEE Computer*, 19(1):50–62, 1986.

[81] Henrik Nottelmann and Umberto Straccia. Information retrieval and machine learning for probabilistic schema matching. *Inf. Process. Manage.*, 43(3):552–576, 2007.

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

[83] Luigi Palopoli, Domenico Saccà, and Domenico Ursino. Semi-automatic semantic discovery of properties from database schemas. In *IDEAS*, pages 244–253, 1998.

[84] Simon Parsons and Anthony Hunter. A review of uncertainty handling formalisms. In *Applications of Uncertainty Formalisms*, pages 8–37, 1998.

[85] Z. Pawlak. *Rough Sets - Theoretical Aspects of Reasoning about Data*. Kluwer Academic, Dordrecht, 1991.

[86] Rachel Pottinger and Philip A. Bernstein. Merging models based on given correspondences. In *VLDB*, pages 826–873, 2003.

[87] Rachel Pottinger and Philip A. Bernstein. Schema merging and mapping creation for relational sources. In *EDBT*, pages 73–84, 2008.

[88] Rachel Pottinger and Alon Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Journal*, 10(2-3):182–198, 2001.

[89] Christoph Quix, David Kensche, and Xiang Li. Generic schema merging. In *CAiSE*, pages 127–141, 2007.

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

[91] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[92] N. Rizopoulos. Automatic discovery of semantic relationships between schema elements. In *ICEIS*, pages 3–8, 2004.

[93] N. Rizopoulos, M. Magnani, P.J. McBrien, and D. Montesi. Uncertainty in semantic schema integration. In *BNCOD'05 (2)*, pages 13–16, 2005.

[94] N. Rizopoulos and P.J. McBrien. A general approach to the generation of conceptual model transformations. In *CAiSE*, volume 3520 of *LNCS*, pages 326–341, 2005.

[95] Nikos Rizopoulos and Peter McBrien. Schema merging based on semantic mappings. In *BNCOD*, pages 193–198, 2009.

[96] Fèlix Saltor, Malú Castellanos, and Manuel García-Solaco. Suitability of data models as canonical models for federated databases. *SIGMOD Record*, 20(4):44–48, 1991.

[97] Glenn Shafer. *A mathematical theory of evidence*. Princeton University Press, 1976.

[98] A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, 22(3):183–236, 1990.

[99] A. Sheth, J. Larson, A. Cornelio, and S. Navathe. A tool for integrating conceptual schemas and user views, 1988.

[100] Guanglei Song, Kang Zhang, and Jun Kong. Model management through graph transformation. *Visual Languages and Human-Centric Computing*, pages 75–82, 2004.

[101] S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, 1994.

[102] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogenous schemas. *VLDB Journal*, 1(1):81–126, 1992.

[103] Robert D. Stevens, Alan J. Robinson, and Carole A. Goble. myGrid: personalised bioinformatics on the information grid. In *ISMB (Supplement of Bioinformatics)*, pages 302–304, 2003.

[104] Mark Stevenson and Mark A. Greenwood. A semantic approach to ie pattern induction. In *ACL*, pages 379–386, 2005.

[105] Michael Stonebraker and Dorothy Moore. *Object-Relational DBMSs: The Next Great Wave*. Morgan Kaufmann, 1996.

[106] Partha Pratim Talukdar, Marie Jacob, Muhammad Salman Mehmood, Koby Crammer, Zachary G. Ives, Fernando Pereira, and Sudipto Guha. Learning to create data-integrating queries. *PVLDB*, 1(1):785–796, 2008.

[107] Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

[108] Guilian Wang, Joseph A. Goguen, Young-Kwang Nam, and Kai Lin. Critical points for interactive schema matching. In *APWeb*, pages 654–664, 2004.

[109] Jiying Wang, Ji-Rong Wen, Frederick H. Lochovsky, and Wei-Ying Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB*, pages 408–419, 2004.

[110] Y. Richard Wang and Stuart E. Madnick. The inter-database instance identification problem in integrating autonomous systems. In *ICDE*, pages 46–55, 1989.

[111] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[112] Li Xu and David W. Embley. Discovering direct and indirect matches for schema elements. In *DASFAA*, pages 39–46, 2003.

[113] L.A. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[114] Lucas Zamboulis, Nigel J. Martin, and Alexandra Poulovassilis. Bioinformatics service reconciliation by heterogeneous schema transformation. In *DILS*, pages 89–104, 2007.