29
Schriften aus der Fakultät Wirtschaftsinformatik und
Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Supporting Format Migration with Ontology Model Comparison

Peter Wullinger

University of Bamberg Press

29 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Band 29

University of Bamberg Press
2018

Supporting Format Migration with Ontology Model Comparison

Peter Wullinger

University of Bamberg Press
2018

Bibliographische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Informationen sind im Internet über http://dnb.d-nb.de/ abrufbar.

Diese Arbeit hat der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg als Dissertation vorgelegen.
1. Gutachter: Prof. Dr. Christoph Schlieder
2. Gutachter: Prof. Dr. Ute Schmid
Tag der mündlichen Prüfung: 22.05.2017

Herstellung und Druck: docupoint, Magdeburg
Umschlaggestaltung: University of Bamberg Press, Larissa Günther
Umschlagbild: © Peter Wullinger

ISSN: 1867-7401
ISBN: 978-3-86309-577-2 (Druckausgabe)
eISBN: 978-3-86309-578-9 (Online-Ausgabe)
URN: urn:nbn:de:bvb:473-opus4-517018
DOI: http://dx.doi.org/10.20378/irbo-51701

Contents

In this thesis, certain (mathematical) notation will be used to represent data structures and operations. An overview can be found in appendix A.1.

Contents i

1. Introduction 1

1.1. Built Heritage Digital Maps 3

1.2. Mapping a Map: Low Effort Migration 5

1.3. Research Questions and Methodology 13

1.4. Structure of this Thesis 16

2. Digital Preservation 19

2.1. Information Representation 19

2.2. Intelligibility 21

2.3. Format Ageing 25

2.4. Structure and Operation of a Digital Archive 26

2.5. Approaches to the Preservation of Digital Documents 29

2.5.1. Intelligibility Management 29

2.5.2. Bitstream Preservation 29

2.5.3. Monitoring Contextual Intelligibility 30

2.5.4. Intelligibility Monitoring 31
2.5.5. Emulation 32
2.5.6. Migration 34
2.6. Significant Properties and Semantic Features 35

3. Format Conversion and Ontology Matching 41
3.1. Ontologies 41
3.1.1. Related Work 42
3.1.2. Ontology Elements 43
3.1.3. Three Views on Ontologies 45
3.1.4. The Ontology 56
3.2. Ontology Alignment 57
3.2.1. Bridging Ontologies 59
3.2.2. Ontology Matching 63
3.2.3. Measuring Alignment Quality 65
3.2.4. Current Developments and Challenges in Ontology Alignment 67
3.3. Document Ontologies 74

4. Automated Reasoning in LillyTab 79
4.1. Knowledge Representation Logics 79
4.1.1. Automated Inference and Reasoning 80
4.1.2. Description Logics 83
4.1.3. Expressive Description Logics 90
4.1.4. Description Logic Knowledge Bases 93

4.2. Automated Reasoning in Description Logics 95
4.2.1. Basics of Tableaux Reasoning 96
4.2.2. Reasoner Optimizations 105
4.3. LillyTab 110
4.3.1. Rationale 112
4.3.2. Tableau Rules 113
4.3.3. Tableau Data Structures 117
4.3.4. Reasoner Implementation 119

5. Completion Graph Based Mapping **121**

5.1. Completion Graphs and Queries 121
5.1.1. Mapping Conjunctive Queries to Completion Graphs . . . 122
5.1.2. Extracting Queries from Completion Graphs 124
5.1.3. Handling Negation 129
5.1.4. Handling Universal Quantifiers 130
5.1.5. Handling TBox-axioms 132
5.1.6. Handling Existential Non-Determinism 134
5.1.7. Benefits of Completion Graph Representation 139
5.2. A Mapping Tableau 139
5.2.1. Extracting the Conjunctive Query 143
5.2.2. Query Complexity 155
5.3. Tuple Generating Dependencies 155
5.3.1. Source-to-Target Property 159

6. Mapping Refinement **163**

6.1. Refinement Process 166

6.1.1. Model-Based Refinement 168

6.1.2. Refinement Strategy 169

6.1.3. Matcher Mode 171

6.2. Refinement Rules 172

6.2.1. Semantic Refinement Rules 175

6.2.2. Subclass Refinement 176

6.2.3. Role Successor Refinement 177

6.2.4. Refinement Performance 178

6.3. Interactive Alignment 184

7. Comparing DL Completion Graphs **189**

7.1. Requirements Analysis 190

7.1.1. Element Level Similarity 190

7.1.2. Maximum Information Transfer and Alignment Length . . 192

7.1.3. Avoiding Unjustified Elements 193

7.1.4. Phrase Extraction 194

7.1.5. Axiom Filtering 200

7.1.6. EGD Path 202

7.1.7. Logical Derivates 207

7.2. Similarity Measures for Completion Graphs 207

7.2.1. Baseline Similarities 207

7.2.2. Governing Term Cosine Similarity 211

7.2.3. Clustered Cosine Similarity 211

7.2.4. Dependency Cluster Similarity 213

8. Evaluation and Conclusion **221**

8.1. Comparing Mapping Rules 221

8.2. Evaluation Design 223

8.3. Evaluation Results 227

8.3.1. Interactive Refinement 227

8.3.2. Effects of Lexical Extractor Parameters 228

8.3.3. Effects of Axiom Set Similarity 229

8.3.4. Effects of Axiom Collection 229

8.3.5. Comparison Of Completion Graph Similarity Measures . . 229

8.4. Discussion 231

8.4.1. Interactive Refinement 231

8.4.2. Automatic Refinement 232

8.5. Summary 236

8.6. Future Work 238

A. Appendix **243**

A.1. Note on Notation 243

A.2. Proofs 247

A.3. Raw Evaluation Results 254

List of Definitions **261**

List of Theorems **265**

List of Figures **267**

List of Tables **271**

List of Algorithms **273**

Index **275**

Bibliography **285**

SECTION 1.0

Introduction

Long term preservation of digital documents is a challenge that is faced by institutions worldwide. Technologies make use of redundant storage and integrity verification to preserve the original raw bitstream of digital documents unaltered. However, while traditional, printed physical documents can be interpreted by domain experts (albeit with some difficulty) even after decades, the same is not true for digital documents. Preserving only the raw bitstream of a digital document does not make it possible to extract the information contained in the document. Applications supporting the document format may no longer run on then-current hardware, new software may use different document formats for various reasons, and documentation on the archived document format may be incomplete or no longer available; in short: being able to read successfully the bits and bytes stored inside an archive does not necessarily mean we are able to extract meaningful information from an archived document.

This problem, addressed in this thesis, is also known as *format ageing*. A document is subject to format ageing if the representation of the document itself is still intact, but its contents cannot be read, because the encoding of the document is no longer known. The problem exists for traditional documents (mostly with texts in ancient languages, e.g. Linear A [Bes72]), but it is digital documents where format ageing is of particular significance, as it happens not at the pace of thousands of years, but within decades. While digital archiving strategies and technology are able to preserve a digital document's exact representation (the bitstream of the document), future applications (or hardware) may not be able to interpret the stored document, essentially rendering the document useless.

One way of solving the format ageing problem is to transform existing documents into more modern document formats whenever the existing document format is threatened to become unreadable. This method requires careful monitoring of available hardware and application software in correlation with an archive's content.

Lowering the effort required to extract a document from a digital archive that is stored in a different format than supported by future applications is the primary focus of this thesis. Finding a viable approach to format ageing problem in domains where only low effort migration projects are possible is a major challenge and more research is needed. This thesis opts for a twofold approach, where part of the work is done at the time of submission of a document into the archive and part of the work is performed in the future when extracting existing documents from the archive and integrating them again into the workflow.

The developed solution utilizes ontological representation of information and techniques from ontology matching to reduce the effort required during the retrieval (dissemination) of a digital document from an archive.

- When documents are stored into the archive, the dataset of the document is converted into an ontological representation format with a well defined ontology schema that is also archived alongside the documents.

- In the future, applications must either be able to work with arbitrary ontological models (and thus support archived ontologies directly), or they must at least support an ontological representation of their internal data model. In the latter case, archived documents need to be converted into the ontology schema supported by future applications, requiring some technique for *ontology alignment*.

Built heritage digital documentation serves as the real world anchor of the developed techniques. Built heritage digital maps are at the same time natural candidates for the ontology-based archiving technique as well as provide a particularly challenging scenario for the long term preservation of domain specific documents.

1.1 Built Heritage Digital Maps

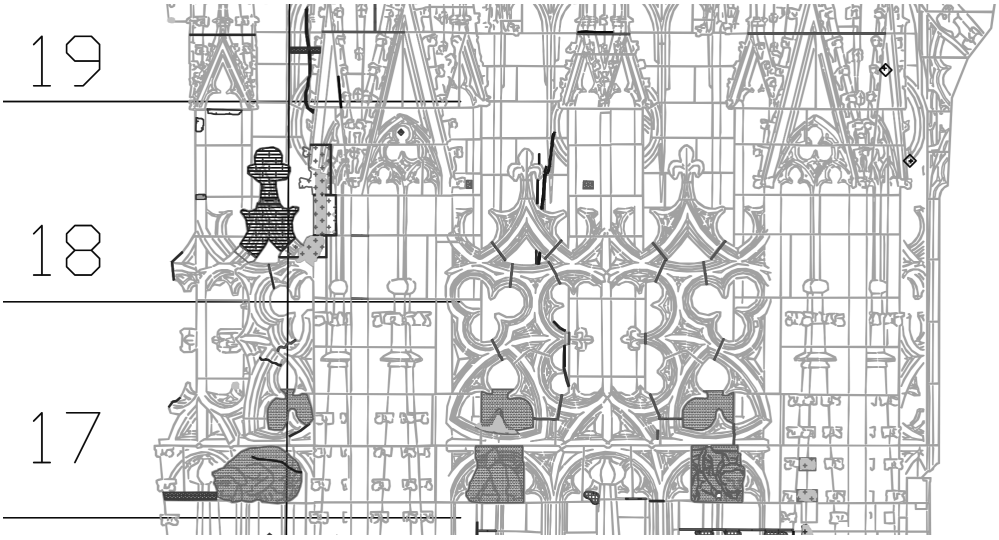
Keeping a modern building in working order is a challenging task. While it may seem static, a building is subject to slow but continual changes and constant maintenance is required. The challenges are even more diverse when the building of interest is several centuries old. Not only are such buildings often in everyday use and must be kept usable, but also the identity of the historic building must be preserved. Ensuring the continued preservation of such historic buildings and monuments requires not only modern technology but also the ability to work with documents and techniques that are often centuries old.

A crucial part in this process is played by documentation. Planning of preservation measures requires detailed knowledge of a preservation site. Not only must the basic layout of a site be documented, but also the specific findings, materials, and techniques originally used. In addition to that, damages must be carefully recorded so that proper planning of suitable preservation measures can be performed. And when they are finally undertaken, the measures themselves have to be documented, because even subtle changes to a building may have consequences unforeseen at the time of execution that show up but decades afterwards.

Most of the data collection and recording task remains manual work. While technological advances have made automatic data collection (e.g. environmental monitoring, monitoring of structural shifts) possible in some cases, assessment of many phenomena is still feasible only by visual inspection and note-taking. Traditionally, such visual inspection was –and often still is– carried out by using a paper plan of the site of interest and coloured crayon. Different colours and/or hatch patterns on such a hand-drawn map indicate different phenomena. The meaning of each colour is documented either directly in a legend on the paper plan or in an accompanying data sheet.

The traditional, “analogue” methods are more and more being supplemented or replaced by computer based digital mapping. The software used for this process is quite varied. Sometimes, only simple drawing or CAD programs are used to essentially re-create a digital version of analogue crayon-sketches as shown in

Figure 1.1. Part of a Built Heritage Digital Map for St. Stephan's Cathedral, Vienna



An outline of the building's exterior features is used as a background map. Findings and measurements are draw in different colors and hatch patterns on top. Large numbers on the left indicate height from the ground.

e.g. fig. 1.1. The visual aspect is very important in the domain and is the focus of many special purpose mapping tools. Some tools, however, recognize the need for structural data acquisition and support more sophisticated data collection. These tools allow to associate additional, structured datasets with each geometric object.

The map documents produced by the respective tools are complex digital documents. Contained within a map document are both a spatial database –often in the form of a proprietary CAD document– as well as a structured dataset –often in the form of a relational database or ontological knowledge base. This complexity together with the particular properties and requirements of the domain of built heritage makes preserving these documents both an interesting task and a demanding challenge.

1. The active lifespan, i.e. the time a document is of direct importance to everyday tasks, is higher for built heritage digital maps than for many other domain documents. While some of the maps may see everyday use, others become only important after a long time of dormancy. For example, plans made shortly after the Second World War may have to be consulted again only now, because the respective part of the building is about to be renovated

or was unexpectedly damaged. On the same line, analogue plans from the early 20th century can become an important source of information for current preservation measures.

2. The exceptionally long timespans of map documents cause a slightly paradoxical situation. While documentation standards in built heritage do not change as fast as in other domains, even the most well defined document format can be expected to fall out of date within the usage period of the respective document. Given the additional fact that some documents may lie dormant in an archive for decades, format migration seems to be almost always a necessary process when an "old" digital document is retrieved from the archive.

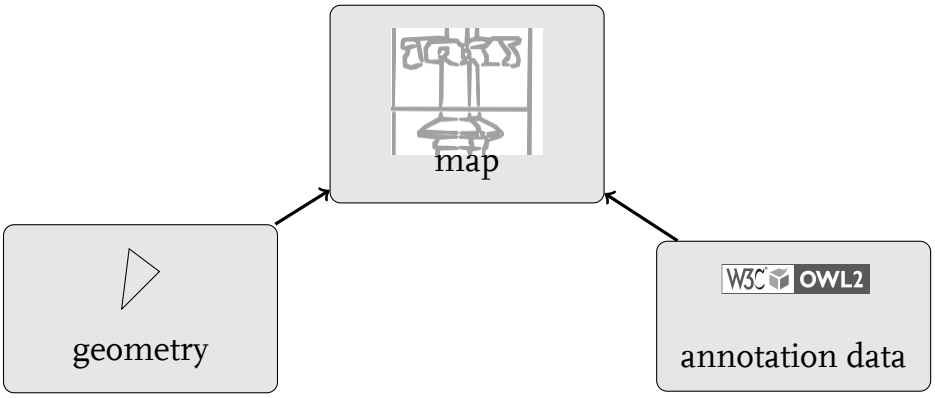
3. Add to this the fact that every larger preservation site usually has its own documentation standards that have often evolved over centuries and are very closely adapted to the ingrained workflows of local preservation scientists and workers. This in turn has lead to the fact that the data values for the recorded phenomena on the digital map (the *data schema*) are slightly different for each historic site even when the buildings are of the same type. We are therefore not only faced with the task of migrating a single data model, but at least one for each large preservation site and/or institution.

1.2 Mapping a Map: Low Effort Migration

In the (not so far) future, the preservation scientist in charge of the digital archive needs to retrieve some digital maps from the archive. These were created twenty years ago and are now needed to aid in the planning of a new, large, and expensive preservation measure.

She is able to retrieve the documents from the archive successfully, but when she fires up the current mapping application of the institution, she is forced to realize, that the map documents from the archive cannot be opened.

Figure 1.2. Structure of an Archived Digital Map in Built Heritage



Because of proper archiving techniques, the individual components of the map have been packaged into a well-known container format. Looking at the individual parts in the container, the preservation scientist notices that the geometry format is still readable and that the structured data and the data schema have been packaged in a well known ontology representation format (figure 1.2).

The new mapping application has an import facility for the geometry format and also for reading data in an ontological document format. The ontological schema, however, differs from the schema of the archived document and the mapping application does not support the old format directly. Fortunately, the ontology schema for the new format is available.

Initial Alignment Our preservation scientist opens both schemas in a standard ontology editor. At a first glance, she sees quite a few similarities and consequently comes to the conclusion, that a translation from the historic format to the new ontology format should be possible. What she wants is a *mapping* from the source ontology into the target ontology. Because she knows that ontology alignment is a well-covered research area, she believes that an *ontology matcher* can be useful.

After the matcher has analyzed both the source and the target ontology, it returns a list of 1:1 correspondences between elements of both ontologies. A partial list of these correspondences can be found in table 1.1. Here, Sandstone (in the his-

Table 1.1. Simple Correspondences for the Example Ontologies

SandStone	$\subset$	Ashlar
LimeStone	$\subset$	Ashlar
Measure — StoneReplacement	$\equiv$	StoneReplacement
Measure — GroutFilling	$\equiv$	Measure — GroutFilling
	$\vdots$	

toric source ontology) is mapped to Ashlar in the modern, target schema as is LimeStone. The subset ($\subset$) relation indicates that Ashlar is the more general type in both bases.

Additionally,

Measure — StoneReplacement and StoneReplacement

are deemed equivalent, as are

Measure — GroutFilling and Measure — GroutFilling.

After some manual corrections and improvements, however, the preservation scientist is forced to realize that the obtained set of mappings is still incomplete. While all primitive correspondences have been established correctly, the mapping does neither cover all available information in the source ontology nor does it make it possible to create any complex structures in the target ontology. For example, the SandStone$_s$[1] and LimeStone$_s$ mappings loose all stone type information.

The preservation scientist realizes that this is a limitation of the ontology matcher, because it does not support complex alignments, but only 1:1 correspondences between ontological elements.

Refinement Because the information from the source documents are desperately needed, the preservation scientist decides to manually derive more expressive mappings. It seems natural for her to use the existing correspondences as a starting point, because the initial alignment is not incorrect, only incomplete.

[1]Concepts from the old/source ontology will be indicated with an $_s$ index. Concepts in the new/target ontology will use a $_t$ index.

Table 1.2. Abbreviations for Concepts and Roles in the Initial Example

MGF	Measure – Grout Filling		
uM	usesMaterial		
Me	Measure	br/Br	brick
Ma	Material	pl/Pl	plaster
GF	Grout Filling	sl/SL	slate
FM	Filling Material	le/Le	lead

She decides to start with the $\mathsf{Measure} - \mathsf{GroutFilling}_s$ concept and writes down the initial correspondence for this concept in first order logic (FOL, [Bar77]):

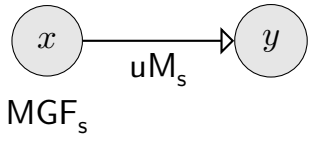
$$\forall\, ?x\,.\,(\underbrace{\mathsf{Measure} - \mathsf{GroutFilling}_s(?x)}_{\text{source side}} \Rightarrow \underbrace{\mathsf{GroutFilling}_t(?x)}_{\text{target side}})$$

Her goal is now to *refine* this initial mapping by adding predicates to both the source and the target terms of the mapping. This makes both sides more distinctive, potentially transferring more information from the source to the target ontology.

The scientist, however, knows that just guessing additional query predicates would not be helpful. Both ontologies are fairly complex and random guessing would neither be efficient nor effective. Instead, she decides that it must be somehow possible to use information from the initial alignments together with the logical axioms of both ontologies to guide her search for *refinements* of the initial mappings.

Luckily, her ontology viewer has support for automated reasoning and allows her not only to show classifications (i.e. relationships between named concepts), but also any additional axioms that hold for individuals of a specific concept.

The reasoner tells her that $\mathsf{Measure} - \mathsf{GroutFilling}_s$ (MGF_s[2]) implies a restriction on the values of the $\mathsf{usesMaterial}_s$ (uM_s) attribute. Our preservation scientist tries to find out more about the possible set of values and creates an instance of the MGF_s concept and attaches a uM_s attribute with an unspecified value to it:



The reasoner, however, only tells her that this construct is consistent and does not give her a list of possible values for uM_s. As a consequence, she is forced to look up the relevant information inside the axiom set of the source ontology. Fortunately, the ontology does not use a complex set of axioms, but instead the values are listed as a simple enumeration (a *dataOneOf* restriction). She finds that only the values $\underline{\mathsf{plaster}}_s$, $\underline{\mathsf{brick}}_s$, $\underline{\mathsf{lead}}_s$, and $\underline{\mathsf{other}}_s$ are allowed as values of the uM_s attribute when attached to a MGF_s instance.

Observing that these seem to be the possible material types for a grout filling, she adds appropriate terms to her initial query, generating a new set of more *refined* query terms that need to be considered individually:

$$\mathsf{MGF}_s(?x), \mathsf{uM}_s(?x, \underline{\mathsf{brick}}_s)$$
$$\mathsf{MGF}_s(?x), \mathsf{uM}_s(?x, \underline{\mathsf{lead}}_s)$$
$$\mathsf{MGF}_s(?x), \mathsf{uM}_s(?x, \underline{\mathsf{plaster}}_s)$$
$$\mathsf{MGF}_s(?x), \mathsf{uM}_s(?x, \underline{\mathsf{other}}_s)$$

[2]The abbreviations from table 1.2 will be used whenever appropriate

She is quite satisfied with the results, because these queries now allow her to distinguish the individual filling types in the source ontology. However, she would have liked more support from her ontology editor and the underlying reasoner. The different values for uM_s could have been suggested by the system, automatically.

Still motivated, the preservation scientist continues to search for more possible refinements, but is unable to find any hints for predicates that she could add to further refine her set of source queries. She also looks at the existing, historic documents and finds that no further attributes are recorded for MGF_s in any of the existing documents and that the queries she has obtained so far seem to cover all relevant information recorded for MGF_s in existing source documents.

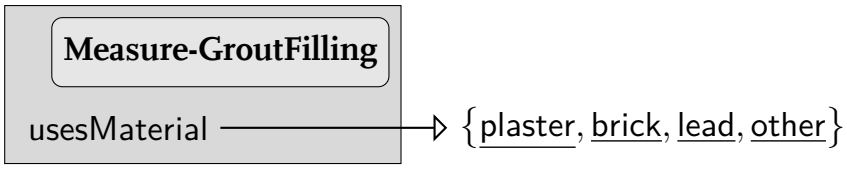
She thus turns her attention to the target ontology. Now, that the source side of her mapping seems to be finished, she needs to find corresponding structures in the target ontology to represent the information extracted by her source queries. Once again, she starts with the initial concept $\mathsf{GroutFilling}_t$ (GF_t) and tries to find any information that is logically implied by the initial concept. Again, she therefore creates an instance of the initial concept and performs a reasoner run, which obtains her the following:

GF_t, Me_t

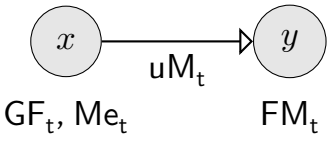
GF_t is thus a subconcept of $\mathsf{Measure}_t$ (Me_t), which re-assures her of the correctness of the initial correspondence, because the pair $(\mathsf{GF}_t, \mathsf{Me}_t)$ fits lexically with the source concept MGF_s.

Thus encouraged, the preservation scientist also observes that GF_t implies again a restriction on some property, this time $\mathsf{usesMaterial}_t$ (uM_t). She assumes that –because of it having the same name– the property is used to represent the same information in the source as well as the target ontology. She then continues with

Figure 1.3. Source Ontology Fragment – Measure — GroutFilling$_s$



the same method that already worked for the source ontology: introducing an unspecified uM$_t$ successor and looking at the implied (i.e. derived by the reasoner) classification. This time, the reasoner returns more information:

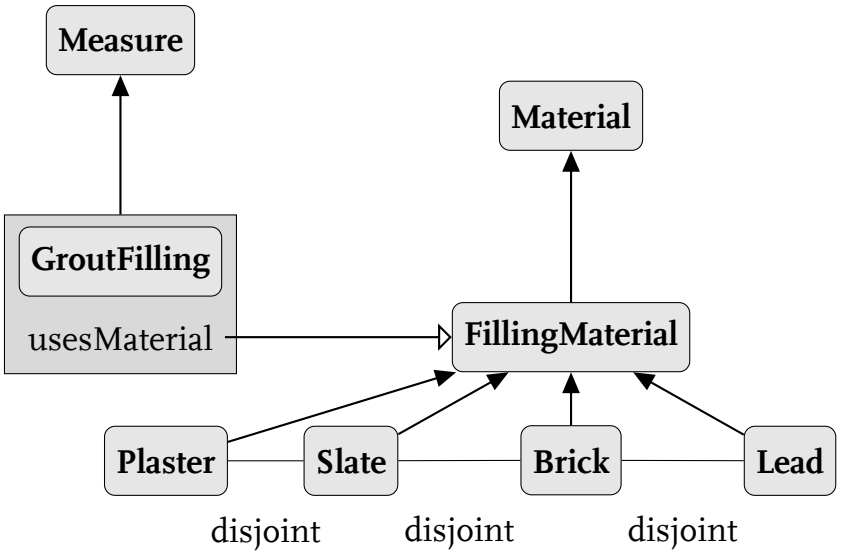


The other end of the uM$_t$ relation needs to be of the type FillingMaterial$_t$ (FM$_t$). After this, however, our preservation scientist is stuck. It is obvious to her, that the individual values of uM$_s$ (in the source ontology) must be mapped onto instances of FM$_t$ (in the target ontology). How to do so, however, is not yet clear to her. FM$_t$ does not imply any restrictions on attributes or relations. No immediate hints within the ontology schema present themselves for further refinement.

A little bit frustrated, the preservation scientist does a search for "Plaster" (one of the source filling material types) in the target ontology. Her search is successful and returns a single concept Plaster$_t$. Even better, the Plaster$_t$ concept is asserted to be a subclass of FillingMaterial$_t$. A closer look at the target ontology reveals that there are four direct subconcepts of FillingMaterial$_t$: Brick$_t$, Plaster$_t$, Lead$_t$, and Slate$_t$. With this information, our preservation scientist is able to draw an overview diagrams of the respective models of the two mapping domains. These are shown in figures 1.3 and 1.4.

It is easy to see, that the modern ontology model is significantly more elaborate, but the depicted fragments have similar information content. Using the information obtained during her manual refinement steps, it is now possible for our preservation scientist to formulate a set of terms that represent the different *refinement steps* she has performed in the target schema:

Figure 1.4. Target Ontology Fragment – $\mathsf{GroutFilling}_t$



$$\mathsf{GF}_t(?x)$$

$$\exists\, ?y \,.\, \mathsf{GF}_t(?x), \mathsf{uM}_t(?x, ?y)$$

$$\exists\, ?y \,.\, \mathsf{GF}_t(?x), \mathsf{uM}_t(?x, ?y), \mathsf{Brick}_t(?y)$$

$$\exists\, ?y \,.\, \mathsf{GF}_t(?x), \mathsf{uM}_t(?x, ?y), \mathsf{Lead}_t(?y)$$

$$\exists\, ?y \,.\, \mathsf{GF}_t(?x), \mathsf{uM}_t(?x, ?y), \mathsf{Plaster}_t(?y)$$

$$\exists\, ?y \,.\, \mathsf{GF}_t(?x), \mathsf{uM}_t(?x, ?y), \mathsf{Slate}_t(?y)$$

Because she does not need a bidirectional mapping, she knows that she does not have to generate a mapping for every target term. Instead, she needs a mapping for every source term, including the initial correspondence.

It is not hard for her to find the correspondences for the $\underline{\mathsf{brick}}_s$, $\underline{\mathsf{lead}}_s$, and $\underline{\mathsf{plaster}}_s$ terms. The representation of the source term with the $\underline{\mathsf{other}}_s$ literal, however, is a problem. The $\underline{\mathsf{other}}_s$ literal has no direct correspondence at the target side. In the source ontology, $\underline{\mathsf{other}}_s$ is a filler for non-existing information rather than

a concrete material type. If viewed in this way, an occurrence of $\underline{\mathsf{other}}_s$ can be interpreted as a non-specified filling material. The same interpretation could have been achieved by simply omitting the property value.

The preservation scientist therefore establishes the following, final refined mapping:

$$
\begin{array}{lll}
\forall\, ?x. & \mathsf{MGF}_s(?x) & \mapsto \mathsf{GF}_t(?x) \\
\forall\, ?x. & \mathsf{MGF}_s(?x), \mathsf{uM}_s(?x, \underline{\mathsf{brick}}_s) & \mapsto \exists\, ?y \,.\, \mathsf{GF}_t(?x), \mathsf{uM}_t(?x, ?y), \mathsf{Brick}(?y) \\
\forall\, ?x. & \mathsf{MGF}_s(?x), \mathsf{uM}_s(?x, \underline{\mathsf{lead}}_s) & \mapsto \exists\, ?y \,.\, \mathsf{GF}_t(?x), \mathsf{uM}_t(?x, ?y), \mathsf{Lead}(?y) \\
\forall\, ?x. & \mathsf{MGF}_s(?x), \mathsf{uM}_s(?x, \underline{\mathsf{plaster}}_s) & \mapsto \exists\, ?y \,.\, \mathsf{GF}_t(?x), \mathsf{uM}_t(?x, ?y), \mathsf{Plaster}(?y) \\
\forall\, ?x. & \mathsf{MGF}_s(?x), \mathsf{uM}_s(?x, \underline{\mathsf{other}}_s) & \mapsto \mathsf{GF}_t(?x)
\end{array}
$$

1.3 Research Questions and Methodology

The introductory example has shown a low scale format migration project. Low scale migrations projects should be expected to be quite common in specialized domains: The ontological scope of the migration is limited to a few, specialized documents and only limited effort can be put into the preservation process.

Assuming the ontological representation is possible with relative ease, this still makes it necessary to perform format migration at various points in the lifecycle of an archived document:

1. during the preparation of documents before submission into the archive,
2. during the maintenance of the archive, as well as
3. during the retrieval (dissemination) of archived documents in the future.

In addition to these challenges, some domains (like built heritage) even impose the requirement that archived documents must not only be intelligible (i.e. via emulation of old hard- and software), but need to be integrated into current workflows, i.e. they must be made usable with future applications.

I have already noted that the intended solution discussed in this thesis is a twofold approach.

- Documents are converted into an ontological representation at the time of archive submission.
- Future applications are assumed to have support ontological representation of datasets, but possibly with a different ontological schema. Mapping of stored document formats into the future schema is done at extraction time.

The example (and this thesis) focusses on the scenario, when format migration is required despite the use of ontological representation systems. This is the case when some future application supports only its own, intrinsic ontological schema. If so, the documents making use of the historic ontology must be *aligned* with the future schema.

This necessity of alignment is also the reason for the choice of an ontological representation. Ontology matching is an important and widely covered research area and consequently, the format migration process can make use of existing technology for ontology matching.

Because of the specialized scenario, existing techniques do not offer sufficient support to simplify or even automate the matching (and alignment) process. The particularities of the scenario form the principal research questions for this thesis:

Derivation of Complex Alignments *Simple alignments* are correspondences between individual ontological elements. These match a concept description in one ontology with a concept description in another ontology, an attribute with an attribute, and a relation with another relation. In the example, simple alignments were used as the starting point. However, simple alignments are insufficient to express the complex mapping rules needed to establish the final, richer alignment from the example.

Current matcher technology is mostly limited to only simple alignments. One goal of this thesis is therefore to *research and develop methods to derive complex correspondences between ontologies.*

Schema Level Alignments with Instance Hinting The mappings generated should be as generic as possible. This thesis therefore opts for a *schema level approach to matching*, because introducing instances of existing documents can make the alignment generalize poorly to other document instances.

However, because the set of existing historic documents can be assumed to be fixed, as no more documents in the "archived format" are generated at the point of map dissemination, the ability to use hints from existing documents to improve the refinement process is also discussed.

Mapping Refinement Ontology matching is already a broadly covered research area. It is therefore expedient to use existing technologies and algorithms as much as possible also for complex alignments. In the work laid out in this thesis, this is done by *refinement*. To derive a complex alignment between two ontologies, the output of an existing ontology matcher is used as a foundation. These initial mappings are successively improved by *refining* them into more complex mapping.

In this thesis, refinement will be developed both as an assistance method for interactive mapping, suggesting individual refinement steps, as well as an *automated alignment system* that outputs a set of suggested mapping rules.

Use of Ontology Semantics Modern ontology matchers make use of automated reasoning to support the mapping task. This makes it possible for matches to avoid inconsistency and incoherence as well as to improve mappings by calculating the logical consequences of already established partial mappings.

Reasoner support can also be put to use for refinement. In this thesis, methods are developed to

1. aid in the representation of complex mapping rules and provide means to determine a *compact representation* of a complex mapping rule;
2. *ensure consistency* of generated complex mappings;
3. *expand value and class enumerations*, when the enumeration is implied by existing axioms in the ontology;

4. *support the refinement process* to ensure that refined mapping rules are *semantically related* to the initial refinements.

While it is possible to implement (1) and (2) using standard reasoning services (subsumption, satisfiability, and consistency testing, see section 4.1.1), (3) and (4) require special purpose reasoning. Such services will also be developed in this thesis.

1.4 Structure of this Thesis

This thesis is divided into eight chapters. Chapter 1 introduces the problem domain, states the principal research questions and gives an overview of the rest of the thesis.

Chapter 2 introduces the concept of digital long term preservation and digital archives. It addresses the challenges that modern, digital documents present with regard to long term preservation and describes the most prominent of the existing solution approaches. It also contains a treatment of *semantic ageing* as a problem of format ageing and intelligibility.

Chapter 3 describes ontologies as modern, formal knowledge representation systems. The text goes on to (section 3.2.2) introduce ontology alignment as a method to resolve semantic heterogeneity between related ontologies and discusses the relevance of ontological methods for format transformation.

Chapter 4 first introduces description logics, a well-known set of formal languages for knowledge representation. It then introduces the "*LillyTab*" reasoner that was developed as part of the thesis to facilitate a new approach to model based refinement for ontology alignment.

Chapters 5 and 6 form the core of this thesis. In chapter 5, the results of LillyTab's reasoning process are used to establish a framework for the *model based* representation of complex ontology mappings between ontologies. In chapter 6, a set of simple rules is established to open up a search space to obtain complex mappings between two ontologies, implementing a suggestion system for refinement rules that can be used for interactive refinement of ontology mappings.

Chapter 7 develops methods to compare DL completions graphs and uses these methods to implement automatic matching. A set of similarity measures is developed, starting from a simple bag-of-words approach and then on to step-by-step enhancement of the similarity calculation with semantic information from both reasoning and the refinement process.

Finally, chapter 8 establishes the evaluation scenario and demonstrates feasibility of the complex ontology alignment method developed in this thesis. The developed method is evaluated using a real world built heritage ontology and the benefits of the semantic integration via tableau reasoning are demonstrated empirically. The chapter ends with a discussion on the benefits and limitations of the proposed method and gives an outlook on future development and remaining challenges.

SECTION 2.0

Digital Preservation

This chapter introduces the concept of representation heterogeneity and its effect on the long term preservation of digital documents in terms of format ageing.

The tasks, architecture, and workflows of a digital archive are described together with current approaches for the long term preservation of digital heritage.

The chapter concludes with an introduction of the concept of semantic preservation as opposed to pure preservation of raw bitstreams.

2.1 Information Representation

Representation of information is an important concept for humans. Not only do we need spoken or written language to communicate with each other, but we also need to represent information for ourselves: for example, if you read a long scientific text, extracting the information and visualizing it is an important method for learning and understanding.

Unfortunately (or fortunately, depending on the point of view), representation of information is not uniform. Consider for example the following question:

What is the connection between the words "eins", "one", "uno", "un", and "en"?

Some of the commonalities are:

- All of these words are written in the same writing system, namely our modern variation of the old Roman alphabet.
- All of these words consist of one or two syllables.

- All of these words have between two and four letters.
- All of these words contain the single letter "n" somewhere in their spelling.

While these similarities between the different words do obviously exist and are proper, they are only superficial. It is intuitively clear, that we are still missing something with regard to the "real" relationship between the presented words, namely the abstract concept behind all the different representations. The above words are simply the formulation of the numeral "1" in different languages (German, English, Italian, French and Danish, respectively). All of the words represent the same abstract concept, but their representation is still different.

What we see here is formally called *semantic heterogeneity* [Ken89]. The semantic content of the listed textual representations is roughly the same (slightly different interpretations in different languages aside), but the representations are different.

To further elaborate on the problem, let us ask another question:

What is the relationship of the word "Μονάδα" to the above words?

Obviously, Μονάδα consists of six letters, does not contain the letter "n", but rather is not written using the Latin alphabet at all. Still, "Μονάδα" is simply the Greek number literal for the number "1" written in the Greek alphabet and thus refers to the same fundamental concept. Here, in addition to using a different language, the underlying alphabet has changed, too. While a reader who knows only the Latin alphabet but none of the languages is able to at least recognize the different letters in the first set of words, she would be completely at a loss when trying to interpret the Greek word.

To enable knowledge sharing, all participants need to a agree on a common representation. As an example at the lowest level, all communication partners need to use the same set of symbols and the interpretation of the symbols must be clear to all of them. Would this text be written in Linear A[3][Bes72], it would be very hard to find a reader (and an author, too).

[3]Linear A is an ancient written language from Crete that dates back to at least 1900 BCE and has not been fully deciphered yet.

That means, that even the present text makes a number of silent assumptions: any interested reader is probably able to decipher the Roman letters. If this is the original text, the reader is also able to read Computer Modern Roman, this being the font used in the initial print layout. A reader is also required to have proficiency in the English language.

At many points within this book, the reader is also assumed to have acquired a significant proficiency in the specialized symbols used to represent mathematical formulæ (although some of them are explained in the appendix).

On top of this, in order to read this book and obtain the information encoded within, the reader must also fulfil additional, higher level requirements. For example, a reader should have fundamental knowledge in computer science in and mathematics beyond the ability to interpret the raw symbols.

2.2 Intelligibility

The property of some representation of information to be available in (or to be convertible to) a format that is accessible to an interpreter (whether human or machine) is called the *intelligibility* [TYK12] of the information. As can be deduced from the observations in the previous section, intelligibility is a principle that is highly specific to both the information to represent and to the interpreter. A German version of this text is useless to a reader (interpreter) that understands only English. The various formulæ, mathematical definitions and derivations later in this text are most likely useless to a reader without mathematical training.

The interpretation process can, however, go wrong also in more subtle ways: printing this text on green paper with red ink possibly makes it unreadable to persons with protanopia[4]. If the interpretation process at some point in time uses the wrong character set, some or all of the symbols in this text maybe misinterpreted and displayed incorrectly.

[4]Protanopia is a medical condition in which the retina has defective or missing detection cones for long (reddish colour) wavelengths. This makes it hard for the affected individuals to distinguish blue and green as well as red and green colour differences.

When using digital documents, the interpretation process is very often automated. For example, while writing this thesis, the computer automatically translates my keystrokes into a binary representation. That binary keystroke is then sent to the character set module and translated into another binary representation so that it represents a particular character. This digital character is sent to the display device manager that looks up that particular character in the character table of a font to get a glyph. And that glyph is displayed to me on screen. Hence, while typing this text, the computer automatically handles more than four different representations of the same piece of information before it is presented back to me in a format that I can digest with my human senses.

Following the nomenclature of Tzitzikas et al. [TYK12], the individual components of the interpretation process are called *modules*. If a module needs another module to function (e.g. a printer is needed to get a printed version of a document and the printer itself needs printer paper) this is called a *dependency*.

Tzitzikas et al. also provide a semi-formal framework to model the interpretation process for digital documents using DataLog [CGT89]. In this framework, the process of interpretation is represented as an n-ary predicate called a *Task*. To determine if a task can be performed, a DataLog query is performed to determine if the task predicate is satisfiable.

We will use typical Prolog-based syntax for DataLog statements. Consequences will be written to the left of the arrow ($\Leftarrow$), while antecedents will be to the right of the arrow. Individual predicates will be separated by "," (indicating conjunction), and statements will be terminated by a trailing dot ".". For example to read this thesis in the print-ready binary form on some reader device, we could define the task "Displayable" as

$$\mathsf{Displayable}\,(Doc, Reader)\,.$$

indicating the operation to display the document Doc on a reader $Reader$. The paper goes on to define a rule-based framework to validate if an interpretation operation is possible by defining (DataLog) rules that represent the interpretation process. For example, to read this document in the Portable Document Format (PDF, [PDF08]), an interpretation rule might look as follows:

$$\mathsf{Displayable}\,(Doc, Reader) \Leftarrow \mathsf{Pdf}(Doc), \mathsf{PdfReader}(Reader).$$

This indicates that to read Doc using a PDF reader, we need both Doc to be in the appointed format and we also need the reading device to be a PdfReader. Of course, the situation is usually more complex than this. For example, this document was not originally encoded in PDF. Rather, in its original form, it is actually a text file with markup that has been *converted* to PDF using a suitable conversion tool. If we try to represent the fact that we also accept formats of this document that can be converted to PDF format using an available tool, we might get something like the following set of rules:

$$\begin{aligned}
\mathsf{Displayable}\,(Doc, Reader) \Leftarrow\ & \mathsf{Pdf}(PdfDoc), \\
& \mathsf{Convert}(Doc, PdfDoc) \\
& \mathsf{PdfReader}(Reader). \\
\mathsf{Convert}(Src, Conv, Tgt) \Leftarrow\ & \mathsf{Converter}(Conv), \\
& \mathsf{CanConvertFrom}(Src, Conv), \\
& \mathsf{CanConvertTo}(Tgt, Conv).
\end{aligned}$$

The potential complexity of representation rules does not stop here, however. It is, for example, possible to take into account the fact that the PDF version of this thesis is generated using the LaTeX document processor. To generate to PDF version, we need a LaTeX processor that supports all the additional packages included in this document:

$$\mathsf{Converter}(\underline{xelatex}).$$

$$\mathsf{SupportsPackage}(\underline{xelatex}, \underline{komascript}).$$

$$\mathsf{CanConvertFrom}(Src, \underline{xelatex}) \Leftarrow \mathsf{LaTeX}(Src), \mathsf{not}(\mathsf{HasMissingPackage}(Conv, Src)).$$

$$\mathsf{HasMissingPackage}(Conv, Src). \Leftarrow \mathsf{not}(\mathsf{NeedsPackage}(Src, Package)).$$

$$\mathsf{HasMissingPackage}(Conv, Src). \Leftarrow \mathsf{LaTeX}(Src), \mathsf{NeedsPackage}(Src, Package), \mathsf{not}(\mathsf{SupportsPackage}(Conv, Package)).$$

This ruleset, however, already has a serious drawback. Since DataLog makes the closed world assumption, not(HasMissingPackage) is always true for LaTeX documents without an explicitly declared package list. Thus missing metadata for a document yields a false sense of security, because with the above ruleset a document would be reported to be Displayable, while in reality it is simply missing necessary metadata. Dropping the first HasMissingPackage rule term is, however, not an option, as there *are* valid LaTeX documents that do not include a package.

It is possible to work around this issue by introducing rules that require documents to explicitly state that all their package dependencies have been declared. This, however, makes the ruleset even more unwieldy:

$$\mathsf{Converter}(\underline{xelatex}).$$

$$\mathsf{SupportsPackage}(\underline{xelatex}, \underline{komascript}).$$

$$\begin{aligned}
\mathsf{CanConvertFrom}(Src, \underline{xelatex}) \Leftarrow\ & \mathsf{LaTeX}(Src),\\
& \mathsf{HasAllPackagesDeclared}(Src),\\
& \mathsf{not}(\mathsf{HasMissingPackage}(Conv, Src)).\\
\mathsf{HasMissingPackage}(Conv, Src). \Leftarrow\ & \mathsf{not}(\mathsf{NeedsPackage}(Src, Package)).\\
\mathsf{HasMissingPackage}(Conv, Src). \Leftarrow\ & \mathsf{LaTeX}(Src),\\
& \mathsf{NeedsPackage}(Src, Package),\\
& \mathsf{not}(\mathsf{SupportsPackage}(Conv, Package)).
\end{aligned}$$

Note that the above is by no means a complete description of all the dependencies of this particular transformation process. It does not take into account that specific version ranges of the LaTeX processor maybe required and it also ignores package versions. LaTeX documents also often heavily depend on other external resources like fonts and the software has an elaborate mechanism of locating these resources [BW97]. Modelling such a complex environment is next to impossible and in the *all or nothing* interpretation of intelligibility, even a single failing module makes a document unreadable. Fully modelling all dependencies of a complex document is a non-trivial effort and care must be taken, to not include rules that yield false conclusions because of missing information.

2.3 Format Ageing

Maintaining readability of digital documents exhibits an interesting duality. On the one hand, it is possible to preserve unmodified copies of digital documents far into the future. Technologies make use of redundant storage and integrity verification to preserve the original raw bitstream of digital documents unaltered.

For the purposes of archiving, the bitstream can be viewed as a static, immutable object and –with some effort– the bit pattern of a document can even be adequately described to allow re-interpretation.

On the other hand, preserving only the raw bit stream of a digital document does not make it possible to extract the information contained in the document. Operating systems, application software, suitable input and output devices are all required to make a document readable. In Tzitzikas' model from the previous section, this is represented by not only the documents themselves having dependencies, but also by the *interpretation modules* being dependent on other (sub-)modules, too.

For example,

- the only application that an archived document was originally created with may no longer run on future hardware or operating systems,
- new software may use different document formats for various reasons,
- documentation on the archived document format may be incomplete or no longer available.

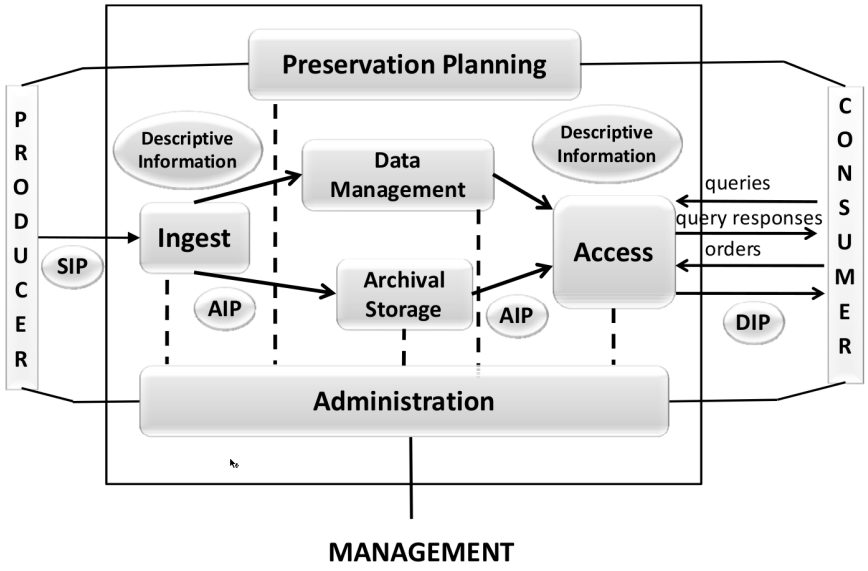
In short: being able to read successfully the bits and bytes stored inside an archive does not necessarily mean we are able to extract meaningful information from an archived document.

Long term preservation remains an open issue with digital documents (see e. g. [Sta05, TB07b, HA07]), because interpretation of documents in undocumented formats cannot be guaranteed.

2.4 Structure and Operation of a Digital Archive

To enable archives to successfully preserve digital documents far into the future, all the concerns that are specific to archiving digital documents have to be identified and suitable solutions have to be provided.

Figure 2.1. OAIS Archive Management Overview



In 2003, the Consultative Committee on Space Data Systems[5] published a first recommendation ("pink book" [fS03]) for a reference model for reliable digital archives, their workflows and requirements. The name of this reference model is Open Archival Information System (OAIS). An updated version ("magenta book" [Con12]) was published in 2012.

OAIS specifies a model for the operation of an archive. At the highest level, OAIS describes five different functional identities (see fig. 2.1) that are involved when a document makes its way into (and back out of) an archive.

Ingest is responsible for accepting data in the form of a *submission information package* (SIP) from a document producer for storage inside the archive. This process is known as *ingest*. During the ingestion process, quality assurance of the submitted package is performed and –if successful– an archival information package (AIP) is generated.

This generation process is one of the most important tasks for the archive. OAIS requires that an AIP provides a "*concise way of referring to a set of information that has, in principle, all the qualities needed for permanent, or indefinite,*

[5]http://www.ccsds.org/

Long Term Preservation [...]"[Con12, p. 4-36]. Or more casually, the AIP is supposed to contain or refer to all information to keep an archived document accessible and intelligible. Added to the AIP is also management information, such as access rights, provenance information, retrieval meta data (termed "finding aids" by the standard), and so on.

Archival Storage is responsible for keeping the bitstream of an AIP available on permanent storage. It also performs many of the tasks of bitstream preservation as outlined later in section 2.5.2.

Data Management handles archive database functions such as indexing and handling database queries.

Administration provides the overall management of the archive, including negotiation of submissions with producers, auditing, configuration management for hard- and software as well as monitoring and other typical management operations of an IT system.

Access is the customer facing part of the archive, providing document search and retrieval as well as access control. Documents from the archive are delivered in the form of a dissemination information package (DIP), which may be derived from one or more AIPs. Note that the OAIS standard allows for the retrieval of excerpts or locally generated statistical data, for example when privacy or copyright concerns do not allow exposing the original document(s).

Preservation Planning "*...provides the services and functions for monitoring the environment of the OAIS, providing recommendations and preservation plans to ensure that the information stored in the OAIS remains accessible to, and understandable by, the Designated Community over the Long Term, even if the original computing environment becomes obsolete ...*"[Con12, p. 4-2].

Intelligibility management is thus a primary concept of preservation planning. It is the responsibility of preservation planning to correlate the contents of the archive with the current technological environment to make sure that all stored documents remain readable.

2.5 Approaches to the Preservation of Digital Documents

2.5.1. Intelligibility Management

A significant problem for the reliable archiving of digital documents is that archived documents are slowly drifting into a state of unreadability. On the bitstream and hardware side this is caused by a process called *silent corruption* [Ros10].

The problem already exists today. With Terabytes and even Petabytes of storage, the likelyhood of a single bit flip in a large data set has become a real problem for storage-heavy applications. Single bit flips often go undetected. A single flip might change a single character in a large document (effectively introducing a typographical error) or cause the colour of a single image pixel to be slightly off. It might also cause a numeric value to change by large magnitudes or make some –especially when compressed– data completely unreadable. Off-the-shelf storage systems have only quite recently been upgraded to detect (and sometimes correct) such bit flips for production storage systems.

2.5.2. Bitstream Preservation

The methods used in a digital archive to protect the original bitstream from change are much the same as in production storage systems: duplication, checksum-based error correction and periodic scrubbing.

Duplication simply refers to the fact that more than one copy of a document is kept. In general, two copies are sufficient to detect an error in a document provided that both documents are compared during dissemination. However, when only the documents are available, at least a third copy must be kept to correct an error that appears in a single document. More copies make the process more reliable, a fact that is mentioned in the moniker of one of the early projects that made use of peer-to-peer networking between different archiving sites to enable duplication of archived content: "*Lot's of copies keep stuff save*" (LOCKSS, [MRG+05]).

Checksumming refers to the practice of creating a binary *digest* of a document, the *checksum*. The technique is already used in many network protocols to detect transmission errors. Because the checksum is evaluated over the bitstream of a message, the checksum also changes when the bitstream is modified. But instead of keeping a full copy for verification, the checksum is usually much smaller. Message digest algorithms developed in the field of cryptography offer even more desirable features: They are meant to be collision resistant, which means that it is hard to find two documents that produce the same checksum. Given a document and its original checksum function, it is possible to determine if a document has been tampered with by checking if the document still hashes to the same digest value. This makes it possible to check the integrity of a document with only a single copy.

Scrubbing Because of silent corruption, it is all but mandatory to perform the consistency check for an archived document frequently. The message digest can only detect a defective document, it cannot repair any damage (this is only possible using forward error correction). Scrubbing is a periodic operation that validates the checksums for all documents. Any damaged document is restored from a copy. As such, damaged documents can be restored before all copies go bad.

2.5.3. Monitoring Contextual Intelligibility

Preserving the bitstream of a digital document is, however, not enough. As described in section 2.2, dissemination of a complex digital document typically requires a large set of *technical dependencies*. If information is stored on special media, hardware must be available to extract data from those media. If specialized software is needed to interpret a document format, the same software or a suitable substitute must be available.

It is important that the dependencies are not only recorded completely at archiving time, but also that the availability of dependencies is closely monitored as time passes. Information technology is a fast moving field and both hardware and soft-

ware become obsolete quickly. This affects the archive, because a document stored in a format that cannot be read with software that runs on existing hardware is effectively lost.

A digital archive is not an isolated entity, but it is dependent on its environment. Special purpose development is possible only in rare cases, most of the time hard- and software to run an archive's infrastructure must be bought on the public market if only for budgetary reasons. The IT market, however, is fast changing and as such archive management must react to this changing environment.

2.5.4. Intelligibility Monitoring

In particular, it is important for the archive management to monitor the intelligibility of its documents. It must track the availability of both hard- and software. However, the situation is more problematic for software than for hardware, because the software ecosystem has significantly faster change cycles. Many large and well-established software vendors release a new version of their software every year and provide support only for a few (e.g. 2 years) previous versions. Support for document formats is usually longer than that (10-15 years) in the sense that newer versions of a software can read old document formats, but even this larger timespan is far from the goal of *indefinite* preservation a digital archive strives to achieve.

The situation in the software world is also more complex, because there is more variation. A well-known document format might be supported by many different implementations. Hence the archive management needs to keep track of different software types for each archived document format.

This need for constant monitoring of the durability of document formats is discussed by Stanescu et al. [Sta05]. The paper develops different criteria for evaluating archiving formats. To store and maintain the results of such an evaluation, Abrams [Abr05] has proposed a global format registry to document which software packages support which document format to what extent. The PRONOM technical

registry[6] provides information about both software (including vendor information and supported file formats) as well as information about file formats themselves, including priority and ancestor/successor information. The global digital format registry (GDFR) failed to produce usable software, but the idea was later inherited by the UDFR project, which has created a "*semantic registry for digital preservation*" using semantic web technology, mostly using PRONOM data.

Unfortunately, the effort of maintaining a format registry has proven to be problematic. Some entries in PRONOM, for example, have not been updated for years, despite clear changes happening in the affected areas. Lawrence et al. already noted that full documentation for file formats, especially for formats of specialized commercial software, is hard to obtain. In addition, this information is also often closely safeguarded by the respective companies.

McGath [McG13] suggests using linked open data and crowd sourcing to update and maintain format registries.

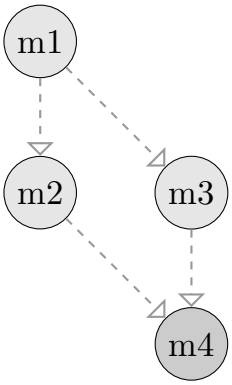
2.5.5. Emulation

Identifying that a document is at risk to become unintelligible is only the first step in the direction of a solution. What if a particular piece of software becomes unavailable, i.e. because the operating system is no longer available for current hardware?

One strategy, proposed by e.g. van der Hoeven [vdHVW05] is *emulation*. Consider figure 2.2, which shows dependencies between four different modules $\mathrm{m1}$, $\mathrm{m2}$, $\mathrm{m3}$, and $\mathrm{m4}$. An arrow between two modules indicates that a module is required by another module. Requirement is transitive, thus $\mathrm{m1}$ requires all modules in the graph and vice versa, all modules depend on the (dark grey) $\mathrm{m4}$. If $\mathrm{m4}$ becomes unavailable, all other (other) modules stop working.

[6] http://www.nationalarchives.gov.uk/PRONOM/

Figure 2.2. Module Dependencies



The general strategy of *emulation* is to replace a failed module (e.g. $\mathrm{m4}$) by a substitute, to *emulate* the interfaces $\mathrm{m4}$ provides to other modules. For example, the *WINE*[7] software package makes it possible to execute many applications that where developed for the Win32 platform on Unix-like platforms. In this case WINE itself is an emulator for the Win32 module.

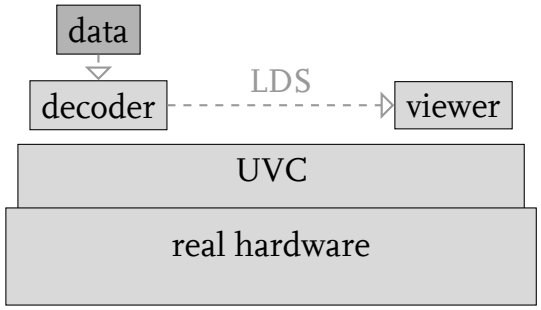
In later research, van der Hoeven [vdHvDvdM05] even goes one step further and proposes to implement emulation already at ingestion time. In the described scenario, the archive only accepts formats that can be interpreted by a well-documented virtual machine, the Universal Virtual Computer (UVC).

The architecture is such that for every document format in the archive, a *decoder* is needed. This decoder needs to be implemented to run on the (virtual) UVC. To simplify this task, the virtual computer specification also provides a *virtual display* and *interaction component* termed a *logical data viewer* (LDV). Communication between the decoder (running on the UVC) and the LDV is performed using a specialized data format, the *logical data schema* (LDS), which –once again– is fully documented.

The effects of this architecture are twofold: ingest costs per data format increase, because for every supported document format, a specialized decoder is needed. Getting sufficient information about a data format is often far from trivial (see section 2.5.4). Also, the effort to write a suitable decoder program for a complex data format is not to be neglected.

[7] http://www.winehq.org/

Figure 2.3. Interpretation with the Universal Virtual Computer



Dissemination and maintenance costs decrease because only the UVC and LDV components need to be supported on the current generation of hardware. The decoder is an UVP program and is stored (and referenced from the AIP of associated documents) inside the archive. Since the stored byte stream of the decoder is a runnable UVC program, it only needs to be extracted and fed to the (future) UVC implementation together with the document.

An implementation of the UVC virtual machine itself and an LDV exists and has been tested in practice using decoders for JPEG, TIFF and GIF formatted images. Partial decoders have also been written for Excel, Lotus 1-2-3 and PDF [LvD05]. None of the decoders expose interactive features of the supported document formats.

2.5.6. Migration

Emulation tries to interpret the originally archived bitstream in its original form. In terms of the module concept, emulation leaves the original document untouched, but affects the dependencies of the document. Another approach suggested by Lorie [Lor00, Lor01] focuses on the original document, instead. The approach recognizes that the semantic content of a document can be represented in multiple different encodings (see also section 2.1) and that the encoding is often replaceable. Hence if a document format is at risk of becoming unreadable, it is possible to re-encode an archived document into a different, more current format while preserving the contained information.

Unfortunately, the complexity of modern document formats usually means that format conversions do not fully preserve the original information. Any complex migration is *lossy* in terms of document semantics. The effect can be easily observed with current software. For example, most office processing software supports import and export in multiple formats. And while the textual content of a document is usually preserved, more complex formatting options often are not. A common occurrence is for embedded graphics to move their position on the page or become distorted (fig. 2.4). In terms of archiving, any transformation that does not re-generate the original user-faced representation of the document is considered *lossy*.

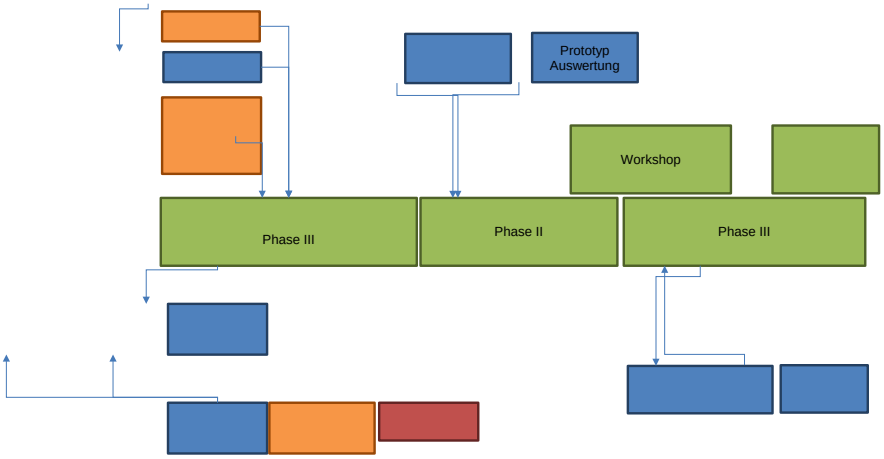
Detecting and measuring semantic loss to a document after a migration is not trivial and limited research is available in the area. Triebsees et al. [TB07a] provide a constraint-based model to measure the preservation of semantic content. In their model, documents or sets of document fragments (for complex digital documents) are represented as graphs. The preservation of semantic content is verified by context-specific invariance checks. They apply their approach mainly to maintain link consistency for HTML documents, but an implementation for verifying the semantic integrity for OpenOffice documents has also been tested successfully. Unfortunately, the approach does not reduce the burden of having to formulate the consistency constraints, which –again– is a non-trivial task for complex document formats.

In addition, the approach requires a sufficiently formal representation of both the source and the target document.

2.6 Significant Properties and Semantic Features

The area of *semantic preservation* is clearly deserving more attention and significant research has been conducted on how to define the semantic content of a document.

Figure 2.4. Distorted Graphics after Import into Different Presentation Software



One of the defining characteristics of complex digital documents is that they are composite documents. This means that every complex document has multiple parts, each with its own unique representation format. For example, built heritage digital maps consist of a geometry document, a document with structured data, and –potentially– often attached documents such as images.

Decomposition of a complex document does not stop there, however. As noted by Hedstrom and Lee [HL02], an even more elaborate decomposition is possible, down to individual, atomic properties, e.g. the colour of a graphical object or the font name of a paragraph. This notion of atomic properties whose appearance inside a digital document can be recorded and identified is important with regard to both dissemination of an unmodified document as well as to format migration.

Dissemination of a digital document can be expressed as an operation with the document. The operation can be as simple as producing a sensory reproduction of the document called an “affordance” [VL04], but the notion also extends to more complex operations like compiling a software application from source code. Hedstrom and Lee limit their interpretation to that of the affordance, but we can assume that this expressed limitation is merely incidental. In general, we can assume that dissemination is performed to facilitate any kind of processing of the archived content, irrespective of the distinction if the processing is performed for human perception or further machine processing.

Hedstrom and Lee further argue that an affordance (or any operation on an archived document) is successful, if the operation reproduces certain properties of the document. For example, if a font specification is missing from an archived text document, a visually identical representation of the text document may not be possible. If a property is required for the success of certain operation, this property is called *significant*. It is important to note that significance is context specific. In the above example, the font is only a requirement, if we need to reproduce the exact same visual representation. If only the textual content of the document needs to be re-produced, the original font is not a significant property as the text itself can be reproduced (potentially in a different font) without knowledge of the font property.

This viewpoint is very much in line with Tzitzikas et al.'s *module* concept. Any operation (which includes viewing) performed with a document is an affordance of that particular document. The parts of the document that are needed to be understood for the operation to succeed can be encoded as dependencies of an "affordance module". The main difference is, that Tzitzikas' model is strongly operational and thus more directly useful.

Providing a suitable definition of a significant property document is open to debate. Hedstrom and Lee give examples of low level properties like basic data unit, byte-level encoding, data type, but also extend their notion to logical data schemata. This notion, of a document property, essentially excludes migration since migration eventually always changes some low level properties of the original document format.

A more elaborate discussion on this topic is given by Yeo [Yeo10]. He makes the important observation that there are both concrete and more abstract properties of a document. For example, there are properties that are content related (e.g. "paragraph 2 contains »A« as first word") and properties that are representation related, such as "The second paragraph has colour $(1.0, 1.0, 0.0)$ in the sRGB colour model". Yeo further argues that content related properties are seen by some as more valuable and that –for many scenarios– reproducing the content related properties is most important.

The concept of significant properties provides a useful modelling tool: It could be argued that an accurate sensory reproduction for some documents might be less important when the document is intended to be part of a future workflow. Instead, here the document's semantics must be reproduced in a way to make the transformed document useable to future software. The compositionality of a (complex) digital document carries over to the semantic space. A document contains atomic "pieces of information" that –combined together– make up the information content of the document itself. Each such semantic information piece comprises a *semantic feature* (definition 2.1).

Definition 2.1 Semantic Feature
A **semantic feature** of a document is an atomic fragment of information independent of its encoding.

The notion of the semantic feature is particularly interesting with regard to format migration. Two documents that represent the same semantic features can for many purposes be considered equal at least with regard to their contextual interpretation. If it is possible to identify and extract such atomic *semantic features* within a document, migrating digital documents becomes a relatively simple, four step process:

1. Identify semantic features in an archived document and select desired features for extraction.

2. Extract the desired semantic features from the archived archived document.

3. Identify the proper encoding of the selected semantic feature in the target document schema.

4. Encode the extracted semantic features into a target document.

In this mode of operation, the abstract model space then contains all *semantic features* can be seen as a *bridge model* between documents with similar semantic content. The drawback of this modelling is, unfortunately, that defining the semantic feature space is not possible in general. However, the concept of an intermediate,

higher level modelling space for documents provides a useful tool to model format migration. Indeed, the concept has been used as a modelling tool for ontology alignments (e.g. [HJK+07]).

In this thesis an attempt is made for a search-based approach to determine semantic features. Starting from an initial feature, we iteratively try to add more semantic features to obtain a *refinement* of the initial feature, that represents more information. This is facilitated using heuristic search, where each search step is meant to represent the introduction of a new semantic feature. By performing this search for both a source and a target document schema (modelled as ontologies), it is possible to search for matching semantic features in both schemas and obtain a set of semantic correspondences between both document schemas.

SECTION 3.0

Format Conversion and Ontology Matching

In the first part of the chapter (section 3.1), we introduce ontologies as a general framework for the representation of information. The chapter provides the definition of formal ontologies as well as a formalization of the basic notions common to most current ontological representation systems.

The second part of the chapter (section 3.2.2) gives an overview of the topic of ontology matching. The chapter ends with a description of some of the remaining research challenges in ontology alignment and a survey of the state of research in addressing these challenges (section 3.2.4).

3.1 Ontologies

Starting in the late 1950s, researchers have tried to lift the problem of knowledge representation onto a higher level of abstraction. Inspired by the human mind, where information seems to be stored as an interlinked network of concepts and typed relationships, semantic networks [Ric56] were developed. At this level, information representation becomes freed from more mundane tasks like the representation of individual data items (e.g. a floating point number). The idea was to enable the (automated) manipulation of knowledge at this higher level without having to consider deeper representation issues.

Formal logic has found its way into knowledge representation in the 1980s. The newly created formal systems made it possible to create frameworks for the exact representation of information. The main benefit of these systems is the ability to derive sound proofs of new propositions from existing background knowledge

using logical reasoning. In many cases, the reasoning process can be automated, giving rise to intelligent knowledge stores capable of answering questions that have been deduced from stored data without the need for explicitly formulating each "known" statement explicitly.

Due to the complexity of the task, however, formal systems are always limited to a specific domain of expertise and even there the stored knowledge is often incomplete. Consequently, most knowledge bases are designed to achieve a particular goal, i.e., they represent abstractions of a particular part of a mental model. The term *ontology* has been adopted to refer both to the abstract mental theory as well as to its machine-readable representation[8]. An ontology is deemed "*an artifact that is designed for a purpose, which is to enable the modeling of knowledge about some domain, real or imagined*" [Gru08]. Ontologies are thus a formal, but restricted and goal-oriented method to represent knowledge only for a certain area of expertise.

3.1.1. Related Work

It is possible to represent knowledge either implicitly or explicitly. Implicit knowledge representation is mainly used for reasons of efficiency and when an explicit formulation is hard to obtain. For example in machine learning, the parameters to describe the minimally separating hyperplane of support vector machines (see e.g. [CST00]) or the signal path weights for neural networks (see e.g. [Mit97]) are an example of implicit knowledge representation.

Explicit knowledge representation is used whenever possible, since it enables the manipulation of the represented information outside the context of individual algorithms. The prevailing notion of an ontology extends to any explicit knowledge representation method. The first type of such description methods were based on the frame and slot metaphor as outlined by Minsky [Min74]. Formal semantics for this *frame*work is given by Brachman [BL84] in first order logic. Kifer and Lausen

[8]This choice of name is not entirely in agreement with the classical, philosophical notion of ontology: in philosophy, ontology describes the area of study that is concerned with being or existence. It is possible to view a (philosophical) ontology as a theory of the nature of existence. In computer science, the emphasis is more focused on the modelling aspect.

[KL89, KLW95] formalize “F-Logic”, a language that aspires to integrate object oriented modelling with relational data storage. F-Logic is still being developed and extended [Kif05, Wei08] and is suggested to serve as the foundation of the Web Service Modelling Language (WSML, [dBFK+05]), a W3C member submission.

All explicit and formal knowledge representation mechanisms have their foundations in formal mathematical logics, with first order logic being the most common base language. Most often, FOL is either extended or the language is restricted to a subset. For example, CycL [LGW88] extends FOL with equality and non-monotonic reasoning. Description logics [BSW02] (see also section 4.1) on the other hand are created through careful bottom-up extension of decidable fragments of FOL.

Multiple formalizations of the concept of an ”ontology” exist. An overview may be found in [Sow00] and partially in [RNC+95]. Our definition is loosely based on the work of Cimiano [Cim06], but I felt free to diverge from that initial definition where convenient. Many other description methods for ontologies focus on the language aspect of the ontology and specify semantics and syntax with regard to a logical language [KS02, FMK+08]. I however, in accordance with [SEH+03], believe that the basic structural properties of an ontology play the same crucial role as their formal, logical semantics.

3.1.2. Ontology Elements

Computer scientists discovered early, that knowledge is fundamentally about the classification of individual objects as well as about the properties and relationships between those objects. This perception of the world follows a philosophical tradition that dates back at least to Aristotle’s metaphysics, where the author describes objects using axiomatic descriptions. In artificial intelligence, even early simple systems like Minsky’s ”frames” group objects into concepts (frames) and allow the definition of attributes and relations (slots). The three basic elements can be found in some form or another in every knowledge representation framework. The principle representation formalism for linked data RDF [LS+99] also supports exactly

the three elements *individuals*, *relations* and *classes*. Most logic-based knowledge representation systems share the same three basic elements with similar semantics.

Instances Instances are the representation of the actual objects of interest, the raw data. An instance may simply be a name for an actual object like my father's car or the number 2, but the notion of an instance also extends to the codification of associations like *These keys belong to my father's car*, that serve to link together other instances.

In some cases, a distinction is made between *individuals* and *data values*.

Concepts Concepts are used to group together similar objects. For example, my father's car and my brother's car are both instances of the concept Car, while my bicycle is not. Yet all three of the mentioned objects would be instances of the concept Vehicle. What is visible here, is, that concepts form a hierarchy: every instance of Car is also naturally an instance of Vehicle, i.e. Car is a *subconcept* of Vehicle. The resulting hierarchy is called a *taxonomy*. The term *class* is often used as a synonym for concept, since concepts provide a *classification* of instances into subsets. Within this work, we do not make a distinction between the two terms.

Relations Being able to describe objects is already expedient, but maybe even more important is, how objects are interrelated and what these interrelations mean. For example, it is obvious, that every lock should have at least one associated key, because the lock would be otherwise unusable. We can capture this fact by creating a relation associatedKey, that defines exactly the key-lock-association. Now, the assertion *These are the keys to my father's car* becomes a relation instance that associates a key object with my father's car.

Many knowledge representation formalisms also allow a restricted arrangement of relations into a hierarchy. For example associatedCarKey would be a *subrelation* of associatedKey such that every instance of associatedCarKey is also an instance of associatedKey.

Definition 3.1 Ontology Elements

The basic elements of a formal ontology O are three, possibly infinite, mutually disjoint sets

- $\mathcal{I}$, the set of **instances**,
- $\mathcal{C}$, the set of **classes**, and
- $\mathcal{R}$, the set of **relations**.

The set of all instances, classes and relations in an ontology O are called the **elements** of O, denoted by $\mathcal{E}$.
In addition to the three basic elements, two additional elements are sometimes mentioned in the literature: The set of attributes $\mathcal{A}$ and the set of attribute types $\mathcal{T}$. The attributes $\mathcal{A}$ are a special relation from individuals $\mathcal{I}$ to types $\mathcal{T}$.
$\mathcal{C}$ and $\mathcal{R}$ are usually enumerable, while $\mathcal{J}$ is sometimes uncountable (e.g. when $\mathcal{T}$ is folded into $\mathcal{I}$ and includes the real numbers).
Attribute values take the form of the extension of types extension(t), $t \in \mathcal{T}$. The values in extension(t) are also called **literals**.
This distinction is not always made and $\mathcal{A}$ is then included in $\mathcal{R}$ and the attribute values are included in $\mathcal{I}$.

3.1.3. Three Views on Ontologies

The principle of interlinked and labelled individuals forming a semantic network or "ontology graph" is the fundamental principle that underlies almost all formal ontological models. However, it is possible to take different points of view with regard to the underlying structure. A different viewpoint does not necessarily mean that the formalism in use is actually a different one, but that the emphasis is on different parts of a particular formalism. I identify three different points of view on formal ontologies, each with its advantages and disadvantages. Each of the viewpoints also forms its own associated research area that has its main focus on the particular viewpoint.

1. The *linked-data view* or *semantic network view* focusses primarily on the structural components of the ontologies. This view is common in the *big data* community, because datasets encountered there are often highly heteroge-

neous and unstructured with very little axiomatization. Consequently, the implicit semantics of the interlinks between objects play the most important role.

2. The most common view presented by ontology editors is the "class centric" or *frame-based* view. This view is inspired by Minsky's [Min74] frame idea: a frame is a collection of attributes and links to other objects. As such, the frame represents the basic data structures of object orientation, where a class is a collection of attributes and relations to other objects. The frame-based view is useful when designing descriptive ontologies, because concepts are usually modelled after real-world objects.

3. Finally, the *logics-based* or *language-based* view describes an ontology as a set of axioms in a logical language L. This has the benefit that the axioms usually have a formal interpretation, which in turn enables automatic reasoning. The formal semantics of almost all modern ontology representation systems are defined by mapping the ontological structures provided by the representation system to axioms in a logical language.

Each view has its specific advantages and disadvantages. The graph based view is sometimes favored because of its interlinking structure and therefore focusses on the links between objects. The frame-based view puts the frame in the center and therefore emphasises the taxonomical structure (and inheritance/subsumption). The language based view, however, is the most semantically expressive one. When the ontological model is mapped directly on top of a formal logical language, the semantic properties of the logical language can be used to perform reasoning within the ontology formalism.

Linked-data View

Graph-centered representation networks like semantic networks [Sow91] focus on the structural components of an ontology: the interrelationship between the elements. The linked-data view is often preferred because of its flexibility and simplicity. The Resource Description Framework [LS+99] has been established as the

de-facto standard for the representation of linked data. In RDF, every piece of information is represented as a directed triple such that every RDF ontology is a labelled graph.

In general, the linked-data view for an ontology can be formalized as an ontology *knowledge base*. The knowledge base is a graph structure that represents individuals and the connections between individuals (definition 3.2).

As mentioned, a knowledge base is the description of the actual instances within an ontology. It is known from mathematical logic, that the actual objects described by the symbols of a logical language are called the *extensions*, of those symbols. The extensional part for ontologies structures is provided by a *knowledge base*.

An ontology knowledge base is similar to the RDF graph representation, however it does not allow for re-ification. When an ontology is observed from a linked-data point of view, the formal semantics of the underlying framework (if any) are usually neglected and the focus is on the connections between data (big data and linked data).

Knowledge bases form a partial order a "lattice of knowledge bases", the *extension lattice*.

Theorem 3.1 Knowledge Base Lattice

$\leq_{\text{ext}}$ is a partial order.

Frame-based View

The frame-based view moves away from the pure connectedness of the graph structure and instead puts a stronger focus on an individual frame. A frame is a (usually named) collection of slots. A slot is either an attribute or a relation schema. Slots can be viewed as the columns of a relational database table or as the fields of a class in an object oriented language.

Definition 3.2 Knowledge Base

Given a set of ontology elements $\mathcal{C}$, $\mathcal{R}$, $\mathcal{I}$, $\mathcal{A}$, $\mathcal{T}$, a **knowledge base**

$$KB \equiv_{\text{def}} (\mathcal{I}, \iota_{\mathcal{C}}, \iota_{\mathcal{R}}, \iota_{\mathcal{A}})$$

consists of

- A set $\mathcal{I}$ of **instance identifiers**, the *universe of interpretation*.
- A relation $\iota_{\mathcal{C}} \subseteq \mathcal{C} \times \mathcal{I}$ called **concept instantiation**.

  $\iota_{\mathcal{C}}$ assigns a set of instance identifiers to each concept $C \in \mathcal{C}$. $\iota_{\mathcal{C}}$ may thus be written as a function $\iota_{\mathcal{C}} : \mathcal{C} \mapsto \mathfrak{P}(\mathcal{I})$.
- A relation $\iota_{\mathcal{R}} \subseteq \mathcal{R} \times \mathcal{I}^+$ called **relation instantiation**.

  $\iota_{\mathcal{R}}$ assigns to each relation a set of tuples, the instances of this relation. $\iota_{\mathcal{R}}$ may thus be written as a function $\iota_{\mathcal{R}} : \mathcal{R} \mapsto \mathfrak{P}(\mathcal{I}^+)$.
- A function $\iota_{\mathcal{A}} : \mathcal{A} \mapsto \mathcal{I} \times \bigcup_{t \in \mathcal{T}} \text{extension}(t)$ called **attribute instantiation**.

$\mathcal{A}$ and $\mathcal{T}$ are often omitted. In this case, $\mathcal{A}$ is assumed to be part of $\mathcal{R}$ and the extensions of the types from $\mathcal{T}$ are assumed to be part of $\mathcal{I}$.
When the type of the argument is clear from the context, we use only ι instead of $\iota_{\mathcal{C}}$, $\iota_{\mathcal{R}}$ or $\iota_{\mathcal{A}}$.

Definition 3.3 Knowledge Base Extension

- Given a knowledge base $KB = (\mathcal{I}, \iota_{\mathcal{C}}, \iota_{\mathcal{R}}, \iota_{\mathcal{A}})$, and a knowledge base $KB' = (\mathcal{I}', \iota'_{\mathcal{C}}, \iota'_{\mathcal{R}}, \iota'_{\mathcal{A}})$ is a knowledge base extension of KB, iff $\mathcal{I}' \supseteq \mathcal{I}$, $\iota_{\mathcal{C}} \subseteq \iota'_{\mathcal{C}}$, $\iota_{\mathcal{R}} \subseteq \iota'_{\mathcal{R}}$, and $\iota_{\mathcal{A}} \subseteq \iota'_{\mathcal{A}}$,
- We write $KB \leq_{\text{ext}} KB'$ iff KB' is an extension of KB.

Instances of a frame need to fill the values of the slots. In the original paper, Minsky [Min74] describes frames as information windows that put a mask on top of the raw data. This mask works like a form, where only certain fields can be filled in. The frame-based point of view can be formalized using the formal concept of an *ontology frame* (definition 3.4).

Definition 3.4 Ontology Frame

An **ontology frame** $\mathcal{OF}$ is an algebraic structure

$$\mathcal{OF} \equiv_{\text{def}} (\mathcal{C}, \leq_{\mathcal{C}}, \mathcal{R}, \leq_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathcal{A}, \sigma_{\mathcal{A}}, \mathcal{T})$$

consisting of

- five disjoint sets $\mathcal{C}$, $\mathcal{R}$, $\mathcal{I}$, $\mathcal{A}$, and $\mathcal{T}$. $\mathcal{C}$ is a set of **concept identifiers**. $\mathcal{R}$ is a set of **relation identifiers**, $\mathcal{A}$ is a set of **attribute identifiers**, $\mathcal{T}$ is a set of **attribute types**. The elements of $\mathcal{C}$, $\mathcal{R}$ and $\mathcal{A}$, $\mathcal{T}$ are called the **elements** of the ontology frame $\mathcal{OF}$.
- $\leq_{\mathcal{C}} \subseteq \mathcal{C} \times \mathcal{C}$ is a partial order between concepts called **concept hierarchy**. $\leq_{\mathcal{C}}$ arranges the set of concepts into a semi-upper lattice.
- $\leq_{\mathcal{R}} \subseteq \mathcal{R} \times \mathcal{R}$ is a partial order between relations. $\leq_{\mathcal{R}}$ is reflexive and transitive, and called **relation hierarchy**.
- $\sigma_{\mathcal{R}} : \mathcal{R} \mapsto C^+$, called the **relation signature** of $\mathcal{O}$. The number $|\sigma_{\mathcal{R}}(r)|$ of concepts referenced from a relation signature is called the arity of $r \in \mathcal{R}$.
- A set $\mathcal{T}$ of **datatypes** such as strings, numbers, etc.
- $\sigma_{\mathcal{A}} : \mathcal{A} \mapsto C \times \mathcal{T}$ is called **attribute signature** and assigns every attribute a type from $\mathcal{T}$ and associates it with a concept from $\mathcal{C}$.

  Attributes are also called *datatype properties*

When it is clear from the context, only σ is used instead of $\sigma_{\mathcal{R}}$ or $\sigma_{\mathcal{A}}$.
Again, $\mathcal{A}$ and $\mathcal{T}$ are often omitted. Once more, if omitted, $\mathcal{A}$ is assumed to be part of $\mathcal{R}$ and the extensions of the types from $\mathcal{T}$ are assumed to be part of $\mathcal{I}$.

The ontology frame as presented here is an extension of the original concept as defined by Minsky [Min74]. Its definition is largely based on the ontological framework developed at the *Institute of Applied Informatics und Formal Description Meth-*

ods (AIFB) at the University of Karlsruhe [SEH+03], where it is defined as a full ontology. However, ontology frameworks usually go beyond an ontology frame and thus a different name is chosen, here.

The ontology frame allows some important observations and definitions that will provide useful for the later discussion.

- Both relations $\leq_{\mathcal{C}}$ and $\leq_{\mathcal{R}}$ are partial orders fulfilling the usual conditions:

$$\forall\, x \,.\, x \leq x \qquad \text{(reflexive)} \tag{3.1}$$

$$\forall\, x, y \,.\, x \leq y \wedge y \leq x \rightarrow x = y \qquad \text{(antisymmetric)} \tag{3.2}$$

$$\forall\, x, y, z \,.\, x \leq y \wedge y \leq z \rightarrow x \leq z \qquad \text{(transitive)} \tag{3.3}$$

  with x, y, z variables from $\mathcal{C}$ or $\mathcal{R}$, respectively.

- $\leq_{\mathcal{C}}$ additionally fulfills the conditions to form $\mathcal{C}$ into a semi-upper lattice:

$$\forall\, x \,.\, x \leq_{\mathcal{C}} \top \qquad \text{(top element)} \tag{3.4}$$

$$\forall\, x, y \,.\, \exists\, z \,.\, \left(\begin{array}{ll} & x \leq_{\mathcal{C}} z \wedge y \leq_{\mathcal{C}} z \\ \wedge & \forall\, w \,.\, \left(\begin{array}{l} x \leq_{\mathcal{C}} w \wedge y \leq_{\mathcal{C}} w \\ \rightarrow z \leq_{\mathcal{C}} w \end{array} \right) \end{array} \right) \qquad \text{(supremum)} \tag{3.5}$$

- $\leq_{\mathcal{R}}$ is a partial order over $\mathcal{R}$. $\leq_{\mathcal{R}}$ is only defined for relations of the same arity, i.e. $r_1 \leq_{\mathcal{R}} r_2$ implies $|\sigma(r_1)| = |\sigma(r_2)|$. $r_1 \leq_{\mathcal{R}} r_2$ orders the various components of the relations into a partial order separately. $r_1 \leq_{\mathcal{R}} r_2$ thus implies $\pi_i(r_1) \leq_{\mathcal{C}} \pi_i(r_2)$ for each $0 \leq i < |\sigma(r_1)|$, where by $\pi_i(r_1)$, we designate the ith component of the tuple r_i.

It is often convenient to reference to the set of all direct and transitive ancestors or successors of concepts and relations within their respective lattices. The notation for these sets has been adopted from a similar notation in [HM00].

Definition 3.5 Sub- and Superclass Set

Given an ontology frame

$$\mathcal{OF} \equiv_{\text{def}} (\mathcal{C}, \leq_{\mathcal{C}}, \mathcal{R}, \leq_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathcal{A}, \sigma_{\mathcal{A}}, \mathcal{T})$$

and a class $C \in \mathcal{C}$, the set $C{\downarrow} \equiv_{\text{def}} \{C' \in \mathcal{C} | C' \leq C\}$ is the **subclass set** (C down) of C.
The set $C{\uparrow} \equiv_{\text{def}} \{C' \in \mathcal{C} | C \leq C'\}$ is the **superclass set** (C up) of C.

Definition 3.6 Sub- and Superproperty Set

Given an ontology frame

$$\mathcal{OF} \equiv_{\text{def}} (\mathcal{C}, \leq_{\mathcal{C}}, \mathcal{R}, \leq_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathcal{A}, \sigma_{\mathcal{A}}, \mathcal{T})$$

and a relation $r \in \mathcal{R}$, the set $r{\downarrow} \equiv_{\text{def}} \{r' \in \mathcal{R} | r' \leq r\}$ is the **subproperty set** (r down) of r.
The set $r{\uparrow} \equiv_{\text{def}} \{r' \in \mathcal{R} | r \leq r'\}$ is the **superproperty set** (r up) of r.

An attribute can only be attached to single concept range and the set of possible values (i.e. the datatype of the attribute) is also fixed by the ontology frame. If $a \in \mathcal{A}$ is an attribute, it may be written as a tuple $(D, R) \in \mathcal{A}$, where D, the *domain* of the attribute is the concept, the attribute is attached to and R, the *range* is the datatype, i.e. the set of possible values for the attribute.

Definition 3.7 Domain and Value Range

For an attribute $a = (D, t) \in \mathcal{A}$, we call $D = \pi_1(a)$ the **domain** and extension(t), with $t = \pi_2(a) \in \mathcal{T}$ the **value range** of a.

In many ontologies, binary relations are the most common form. If only binary relations are allowed, i.e. $\sigma_{\mathcal{R}} \subseteq \mathcal{C} \times \mathcal{C}$, the expressive power of an ontological structure is not reduced. Any n-ary relation can be decomposed into a set of binary relations and it always possible to reconstruct the original n-ary relation (see e.g. [JS91, Par02]). Because binary relations are the most common and often the only relations allowed in many concrete knowledge representation systems, they are given a separate name.

Definition 3.8 Role
Given an ontological frame $\mathcal{OF}$ and a relation r, we call r a **role**, iff $|\sigma(r)| = 2$.
r is also called an **object property**.

A role has a natural sense of direction. It is possible to write a role r as a pair $(D, R) \in \mathcal{C} \times \mathcal{C}$, where the role reaches from D to R. In this case, only individuals that are instances of D are allowed at the start of the role arrow and only individuals that are instances of R are allowed at the end a role arrow. Similar to attributes, D and R are called the *domain* and *range* of r:

Definition 3.9 Domain and Range
For a role $r = (D, R) \in \mathcal{R}$, we call $D = \pi_1(r)$ the **domain** of r and $R = \pi_2(r)$ the **range** of r.

Because the frame-focussed view allows it to view an ontology from the point of a particular concept's instance ("my attributes", "my relationships to other instances"), it is the approach taken on by many ontology data editors by creating one data entry dialog per concept. Additionally, the frame-based view has a direct representation in object oriented programming (classes, fields, and references) and database modelling (tables, attributes, foreign keys).

The frame-focussed view, however, yields no provision to specify more complex rules and restrictions beyond inheritance, role domains and ranges.

Language-based view

The last restriction from the previous section is lifted in the language-based view. In the language based view, the constraining axioms and assertions are described in a logical language. Then again, the semantics of the logical language are defined using either first order logic (FOL) or in some higher order logic itself. This makes it possible to perform automated consistency checking when a deduction system for the underlying logical language is available.

All modern formal knowledge representation frameworks map their ontological structures into a logical language and thus define their formal semantics in terms of the logical language. In addition to providing a formal interpretation, logical languages often also provide the ability to formulate much more complex constraints beyond what is available in other frameworks.

The linked-data view is essentially an unconstrained graph of concept assertions and links between individuals. The ontology frame provides the limited ability to constrain the domain and ranges of roles and attributes. While the constraints provided by the frame are often sufficient to describe the knowledge structure of a particular domain, complex interdependencies cannot be expressed in the frame structure alone. This gap is filled by the language-based view, where axioms in the logical language can be used to express general constraints that hold within an ontology.

The logical split between *global axioms* that specify global constraints that hold for any individual in the ontology and assertions that are made about discrete individuals is very often made explicit. An ontology in a logical language is often specified as the union of two disjoint axiom sets: the set of terminological axioms (TBox) and the set of assertional axioms (ABox).

ABox Assertional knowledge in the ABox encodes the information that is considered part of the knowledge base of an ontology (see definition 3.2). An ABox contains information about concrete individuals and concrete relations in the form of *concept assertions* and *property assertions*.

concept assertion A concept assertion assigns an individual x to a concept C. Concept assertions are usually written as $C(x)$ or $(x : C)$.

property assertions A property assertion defines a connection between an individual x and an individual (for roles) or literal (for attributes). Concept assertions are usually written as $r(x, y)$, asserting an r-link between x and y.

TBox While assertions in the ABox refer to discrete individuals, terminological assertions in the TBox are valid for all individuals in the ontology. In many less expressive logics, TBox axioms are unable to refer to discrete individuals at all.

The axioms in the TBox restrict the set of "allowed" (consistent) knowledge bases. Knowledge bases that "fit" (are consistent with) a TBox must adhere to certain restrictions that are imposed by the interpretation of the TBox axioms. For example, restrictions like the domain and range constraints in the frame-based view are usually encoded as TBox axioms.

Typically, the axioms inside the TBox are formulated in a logical language L. Within the language, a (possible infinite) set of axioms can be expressed. The set of all of these axioms form the *signature* Σ_L of the language (definition 3.10).

Since L must refer to the elements of the ontology, L is specific to the element sets $\mathcal{I}$, $\mathcal{C}$, $\mathcal{R}$ of the underlying ontology. It is thus formally necessary to refer to $L(\mathcal{I}, \mathcal{C}, \mathcal{R})$ or even $L(\mathcal{I}, \mathcal{R}, \mathcal{C}, \mathcal{A}, \mathcal{T})$

As usual, we will omit the parameters when the type of L is clear from the context.

Definition 3.10 Language Signature

Given a language L, the signature Σ_L of the language is the set of all sentences that can be formulated in L.
If L is a logical language Σ_L is the set of all possible axioms from L.

When no concrete language for a TBox is given, we express TBox-formulæ in first order logic, e.g. $(\forall\, x\,.\, C(x) \Rightarrow D(x)) \in \text{TBox}$.

But even without resolving to a concrete axiom language and interpretation, it is possible to define the TBox as a constraint system over the set of concept, relation, and attribute instantiations.

When a knowledge base is not a subset of an OCS, it contains a contradiction (according to the semantics defined by the OCS itself). An OCS is an abstract method to define the set of *consistent* knowledge bases. The set of consistent knowledge bases is defined by explicit enumeration.

Definition 3.11 Ontology Constraint System

An **ontology constraint system** OCS is a set

$$OCS \subseteq \mathfrak{P}\left(\mathcal{C} \times \mathcal{I} \times \mathcal{R} \times \mathcal{I}^{+} \times \mathcal{A} \times \mathcal{T}\right)$$

where the constituents are defined in the same way as for ontological frames (see definition 3.4) and knowledge bases (see definition 3.2):

- $\mathcal{I}$ is a set of instance identifiers.
- $\mathcal{C}$ is a set of concept identifiers.
- $\mathcal{R}$ is a set of relation identifiers.
- $\mathcal{A}$ is a set of attribute identifiers.
- $\mathcal{T}$ is a set of types, with extension$(t), t \in \mathcal{T}$ representing possible attribute values. Attribute values are often also called **literals**.

Definition 3.12 Consistent Knowledge Base for an Ontology Constraint System

A knowledge base $KB = (\mathcal{I}, \iota_{\mathcal{C}}, \iota_{\mathcal{R}}, \iota_{\mathcal{A}})$ is **consistent with regard to an ontology constraint system** OCS iff there is an extension $KB' = (\mathcal{I}', \iota'_{\mathcal{C}}, \iota'_{\mathcal{R}}, \iota'_{\mathcal{A}})$ of KB, that

1. that has non-empty interpretations for all elements, i.e. $\forall\, C \in \mathcal{C}\,.\,\iota'_{\mathcal{C}}(C) \neq \emptyset$, $\forall\, r \in \mathcal{R}\,.\,\iota'_{\mathcal{R}}(r) \neq \emptyset$, and $\forall\, a \in \mathcal{A}\,.\,\iota'_{\mathcal{A}}(a) \neq \emptyset$.
2. $\iota_{\mathcal{C}} \times \iota_{\mathcal{R}} \times \iota_{\mathcal{A}} \in OCS$

We write $OCS \models KB$ if KB is consistent with OCS.

This definition allows for *partial knowledge bases*, where only a subset of relevant information is available. Note, that our definition of consistency is compatible with both the closed and open world assumption [Rei87].

3.1.4. The Ontology

The preferred viewpoint for an ontology depends on the respective scenario. "Big data" applications tend to focus on the linked data view. Ontology editors often show a frame-based view of an ontology to reduce information overload at the side of the user. Most reasoning systems have a hybrid approach between the language based and the linked data view (e.g. tableau reasoning for description logics, see section 4.2).

Most ontology formalisms define a mapping between the different viewpoints. For example, both versions of the Web Ontology Language [MvH04, MPSH08] have a direct mapping into the Resource Description Framework (RDF, [LS+99]). The OWL/XML representation of OWL2 [PPSM09], however, focusses more on a frame-based view and partially supports higher level axioms. It is possible to find such correspondences for most ontology representation frameworks and thus the three views on ontologies presented above are indeed only views and not representation formalisms in their on right. Consequently, a definition of an *ontology* itself is possible without resolving to a particular viewpoint, but instead by combining all viewpoints:

Definition 3.13 Ontology

An **ontology** O is a structure

$$O \equiv_{\text{def}} (OCS, KB)$$

consisting of

- an ontology constraint system OCS (described by a set of TBox axioms), and
- a knowledge base KB (described by a set of ABox assertions).

In this thesis the terms “knowledge base” and ABox will be used interchangeably. Most of the time, the term TBox will be used instead of the –slightly unwieldy– “ontology constraint system”.

With the definition of an ontology established, it is also possible to define the consistency (definition 3.14) and inferability (definition 3.15) of an axiom with regard to/from an ontology. For both semantic relations we commonly use shortcut notation. For example, if O is an ontology with language L_O, β is an axiom over a language L_β, and ϕ is an axiom from $L_O \cup L_\beta$, we write $O \cup \beta \models \psi$. Formally, this requires extending the ontology with β similar to what is done in definition 3.14 and then checking if ϕ is inferable from the extended ontology.

Definition 3.14 Consistent Axiom

Given an ontology $O = (\text{TBox}, KB)$ over an ontology language L, and the knowledge base $KB_\phi = \left(\mathcal{I}_\phi, \iota_{\mathcal{C}_\phi}, \iota_{\mathcal{R}_\phi}\right)$ with

- sets of ontology elements $\mathcal{C}$, $\mathcal{R}$, $\mathcal{I}$, and $\mathcal{A}$,
- a formula $\phi \in \Sigma_L$
- $KB_\phi = \left(\mathcal{I}_\phi, \iota_{\mathcal{C}_\phi}, \iota_{\mathcal{R}_\phi}, \iota_{\mathcal{A}_\phi}\right)$,
- x, a fresh individual such that $\mathcal{I}_\phi = \mathcal{I} \cup \{x\}$ so that KB_ϕ is an extension of KB with that individual,
- C_ϕ, a fresh concept and $\mathcal{C}_\phi = \mathcal{C} \cup \left\{C_\phi\right\}$ so that, again, KB_ϕ is an extension of KB with that concept,
- $\mathcal{R}_\phi = \mathcal{R}$, $\mathcal{A}_\phi = \mathcal{A}$, and
- $\text{TBox}_\phi = \text{TBox} \cup \left\{\left(C_\phi \Rightarrow \phi\right)\right\}$, so that TBox_ϕ is an extension of TBox,

ϕ is **consistent** with regard to O, iff KB_ϕ is consistent with regard to TBox_ϕ.

3.2 Ontology Alignment

As already noted in section 2.1, any encoding of information faces the problem of semantic heterogeneity. Similar pieces of information can be encoded in a variety of different ways. The problem also persists with ontological modelling. It is a common scenario for different domain experts to independently design ontology

Definition 3.15 Inferable Axiom

Given an ontology $O = (\text{TBox}, KB)$ over an ontology language L, and the knowledge base $KB_{\neg}\phi = \left(\mathcal{I}_{\neg\phi}, \iota_{\mathcal{C}_{\neg\phi}}, \iota_{\mathcal{R}_{\neg\phi}}\right)$

- sets of ontology elements $\mathcal{C}$, $\mathcal{R}$, $\mathcal{I}$, and $\mathcal{A}$,
- a formula $\phi \in \Sigma_L$
- $KB_{\neg\phi} = \left(\mathcal{I}_{\neg\phi}, \iota_{\mathcal{C}_{\neg\phi}}, \iota_{\mathcal{R}_{\neg\phi}}, \iota_{\mathcal{A}_{\neg\phi}}\right)$,
- x, a fresh individual and $\mathcal{I}_{\neg\phi} = \mathcal{I} \cup \{x\}$ so that $KB_{\neg\phi}$ is an extension of KB with that individual,
- $C_{\neg\phi}$, a fresh concept and $\mathcal{C}_{\neg\phi} = \mathcal{C} \cup \left\{C_{\neg\phi}\right\}$ so that, again, $KB_{\neg\phi}$ is an extension of KB with that concept,
- $\mathcal{R}_{\neg\phi} = \mathcal{R}$, $\mathcal{A}_{\neg\phi} = \mathcal{A}$, and
- $\text{TBox}_{\neg\phi} = \text{TBox} \cup \left\{\left(C_\phi \Rightarrow \neg\phi\right)\right\}$, so that $\text{TBox}_{\neg\phi}$ is an extension of TBox.

ϕ is **inferable** from O, iff $KB_{\neg\phi}$ is inconsistent with regard to $\text{TBox}_{\neg\phi}$.
We write $O \models \phi$ or even $KB \models \phi$ when the remainder of O is clear from the context.

schemas for similar problem domains and then for the resulting models to differ substantially. This in turn originated a research discipline that concerns itself with overcoming semantic heterogeneity between ontologies: *ontology alignment*.

In the literature, the term itself is used in an overloaded fashion. In particular, *ontology alignment* may refer to at least three related, but distinct concepts:

- In a high level context, ontology alignment is the research area that concerns itself with overcoming semantic heterogeneity between ontologies. This is the most abstract use of the term.
- In a more specific usage, ontology alignment refers to *the process of determining correspondences between two or more ontologies*. Because of the possibility of confusion, this process will be referred to as the *ontology alignment process* or *ontology matching* within this thesis.
- Finally, the alignment process produces a result in the form of correspondences between ontologies. This result itself is also called an *alignment*.

This chapter will touch all three different interpretations.

3.2.1. Bridging Ontologies

The basic idea of ontology alignment is similar to the concept of semantic features as described in section 2.6: similar (semantic) fragments of information can have different encodings. In ontological terms, a knowledge base fragment $KB_{1,i} \subseteq KB_1$ from one ontology $O_1 = (OCS_1, KB_1)$ can represent the same information as another knowledge base fragment $KB_{2,j} \subseteq KB_2$ from another ontology $O_2 = (OCS_2, KB_2)$. The goal of an ontology alignment system is to detect and express such correspondences so that matching ontology fragments can be exchanged from O_1 to O_2 and vice versa.

Euzenat and Shvaiko [Euz07] give a basic model of the operation of an ontology alignment system. The model shown in fig. 3.1 extends their model. A matching system takes as input one or more ontologies (with possibly empty knowledge bases) and produces a set of *alignment elements*. An alignment element selects a source knowledge base fragment $KB_{s,i}$, consistent with some ontology O_s and transforms it into a target knowledge base fragment $KB_{t,i}$, consistent with some ontology O_t.

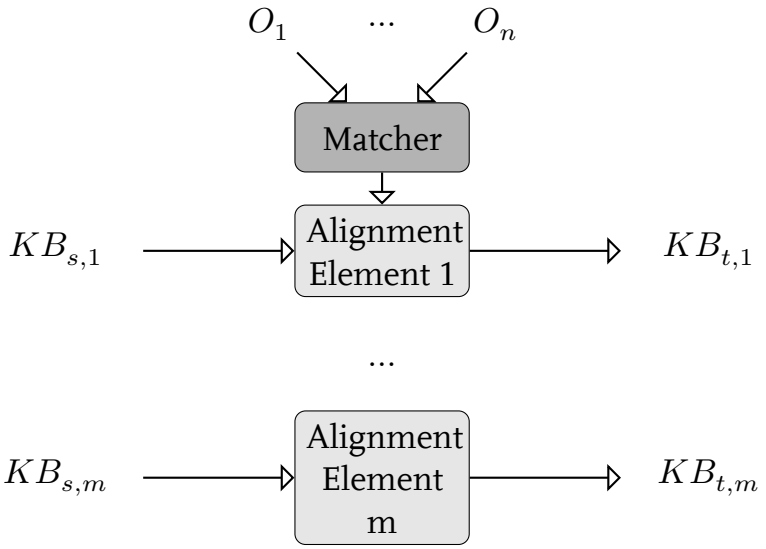
Most matchers work pairwise, that is, the matcher accepts a pair (O_s, O_t) of ontologies and produces a set of alignment elements that transfer instances from KB_s to KB_t.

In principle, it is possible for an alignment element to be imperative. In this case, the matcher might be a code generator producing imperative code in a programming language. Most matchers, however, produce declarative alignment elements. Such a declarative alignment element is defined by Euzenat [Euz04] in terms of a *correspondence* (definition 3.16)

Correspondences can have various levels of complexity. One possible categorization is given by Euzenat [Euz04]:

Level 0 Both entities e and e' need to be atomic elements (concepts, roles, individuals) of the respective ontologies and the degree of confidence n is a floating point value from the interval $[0; 1]$

Figure 3.1. Alignment Generation



Definition 3.16 Alignment Correspondence

An **alignment correspondence** is a 4-tuple (e, e', R, n), with

- e, an **entity** or **cell**, specifying the first endpoint of the correspondence.
- e', another **entity** or **cell**, specifying the second endpoint of the correspondence.
- R, a **relation** that holds between e and e', asserted by this correspondence.
- n, the **degree of confidence** or **strength** of the correspondence between e and e'.

  The maximum confidence is $\top$ and the minimum confidence is $\bot$.

Definition 3.17 Alignment

An **alignment** $\mathbb{A}$ is a collection of *alignment correspondences* $\{c_0, \ldots, c_n\}$.

A level 0 alignment with both $R = \text{"} = \text{"}$ and $n = \top$ is called a *simple alignment* 3.18.

Level 1 Entities e and e' are lists of elements.

Level 2 In the most expressive form of alignments, correspondences take the form of formulæ in some alignment language L_b.

If L_s is the language of ontology O_s and L_t is the language ontology O_t, level 2 correspondences are directional and can be expressed as clauses of the form

$$\beta \equiv_{\text{def}} \forall X \,.\, (\phi(X) \rightarrow \exists Y \,.\, \psi(X \cup Y))$$

where $\phi \in \Sigma(L_s) \cup \Sigma(L_b)$ and $\psi \in \Sigma(L_t) \cup \Sigma(L_b)$. ϕ is thus a formula over the source ontology, potentially enhanced with additional language constructs from the *bridging language* L_b and ψ is a formula over the target ontology, again with additional constructs from L_b. Both ϕ and ψ contain variables from X or $X \cup Y$, respectively.

Informally, the left side of β is interpreted over the source ontology O_s and the right side is interpreted over the target ontology O_t, while β creates a semantic relation between both ontologies.

β is also known as a *bridge rule* (definition 3.19)

Definition 3.18 Simple Alignment

An alignment $\mathbb{A}$ (definition 3.17) is called **simple** iff for all alignment elements $c \in A, c = (e, e', R, n)$

- c is a level 0 element, i.e. e and e' are atomic elements of their respective ontologies,
- $R = \text{"} = \text{"}$, and
- $n = \top$.

The advantage of using declarative correspondences is that their semantics can be defined formally. In addition, level 0 and level 1 correspondences are independent of the respective ontology language.

Definition 3.19 Bridge Rule

Given two ontologies O_s with language L_s and O_t with language L_t a formula

$$\beta \equiv_{\text{def}} \forall X \,.\, (\phi(X) \rightarrow \exists Y \,.\, \psi(X \cup Y))$$

is called a **bridge rule** iff

- $\phi \in \Sigma(L_s) \cup \Sigma(L_b)$.

  ϕ is a formula over the source ontology combined with the bridge language L_b as a function of free variables X.

- $\psi \in \Sigma(L_t) \cup \Sigma(L_b)$

  ψ is a formula over the target ontology combined with the bridge language L_b as a function of variables $X \cup Y$. The variables in Y are also called **generating**.

Care must be taken when evaluating level 2 alignments. Because arbitrary formulæ are supported, the (internal) semantics of both of the source ontology or the target ontology might change when a bridge rule is introduced. Introducing a bridge rule *can potentially invalidate existing information*. More formally, some knowledge base KB_s (KB_t) that is consistent with OCS_s (OCS_t) might become inconsistent when a bridge rule β is added. This behavior is undesirable from a bridge rule, as it should only transfer information from one ontology to the other.

Mapping rules that do not affect the semantics of the ontology constraint systems they link are called *conservative bridge rules*.

Definition 3.20 Conservative Bridge Rule

A bridge rule (definition 3.19) β is called a **conservative bridge rule** iff for the ontologies O_s and O_t bridged by β,

$$\forall \phi \,.\, O_s \cup O_t \models \phi \Leftrightarrow O_s \cup O_t \cup \beta \models \phi$$

for all axioms $\phi \in \Sigma_{L_s} \cup \Sigma_{L_t}$.

3.2.2. Ontology Matching

While bridge rules can be formulated manually, this is a very tedious and error prone endeavour and tool support is all but mandatory. Automating or providing support in creating bridge rules is the job of an *ontology matching* system. The task of such a matcher is relatively easy to describe: take as input one or more ontologies and produce alignments between these ontologies.

The simple description is in contrast to the actual complexity of the task, which has been compared to the complexity of automated language translation. The comparison seems warranted, turning ontology matching into an AI-complete [GS08] problem domain.

Ontology matching is both a well-covered but also evolving research area. In fact, Otero-Cerdeira et al. [OCRMGR15] note "*The amount of research papers published nowadays related to ontology matching is remarkable and we believe that reflects the growing interest of the research community. However, for new practitioners that approach the field, this amount of information might seem overwhelming.*" [OCRMGR15, p. 1]. In their survey paper the quoted authors list as many as 60 different matcher systems that have been developed since 2003. Existing systems take a variety of approaches to the matching task. Euzenat and Shvaiko [Euz07] provide a two-axis classification system (fig. 3.2), distinguishing matchers based on

- how systems process and interpret input data and
- which type of processing or interpretation systems use internally.

The result is a classification system with nine different concrete categories, where many systems fall into one or more of the supplied categories because they combine multiple techniques.

The result is a –slightly chaotic– *zoo* of matching systems. In addition to this, table 3.1 also shows that the quality of generated alignments depends on the structure of the underlying ontology. This means that not every matching system is suited to every alignment task. For example, trying to obtain simple alignments

Figure 3.2. Ontology Matching System Classification (figure from [Euz07])

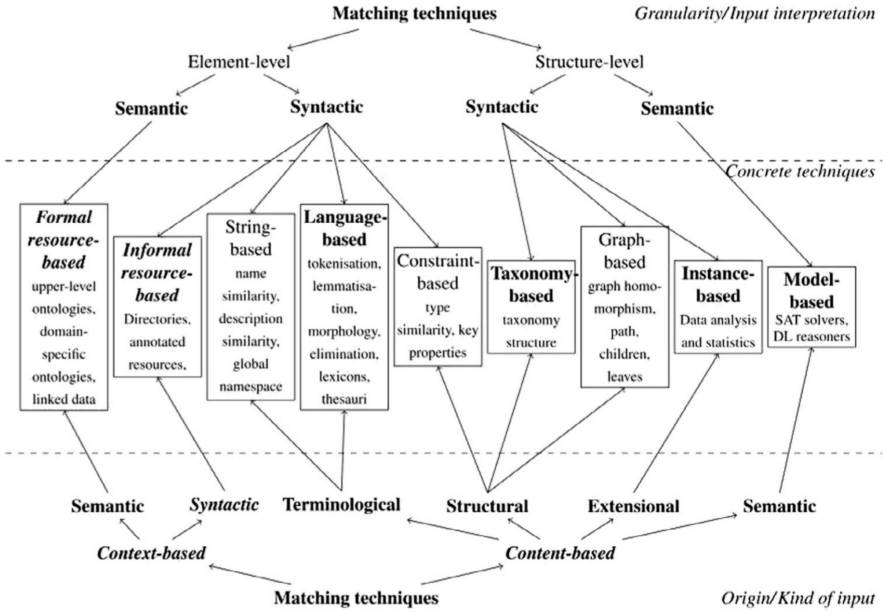


Table 3.1. Simple Matching Results for the "Dom Bamberg" Ontology and Its Evolved Successor

matcher	#	prec. %	rec. %	f %
1. vector-based multi-words	49	55.1	26.7	36.0
2. lexical synonyms	39	59.7	22.8	32.9
3. parametric string	53	54.7	28.7	37.7
4. similarity flooding	19	9.1	1.0	1.8
5. AnchorFlood	47	42.6	19.8	27.0
6. AROMA	40	67.6	24.8	36.2
7. Blooms	12	41.7	5.0	8.8

to map the ontology of the Bamberg Cathedral to a newer version shows that matching precision fluctuates between 9.1% and 67.6% depending on the employed matcher (table 3.1).

Determining a-priori if a matcher is suitable for a particular matching task is usually only possible superficially. Structural matchers show their potential, when two ontologies are structurally similar but are in two different languages. Lexical matchers are good when used in a narrow domain with a well-defined nomencla-

ture (such as built heritage). Some matchers try to overcome these problems by combining multiple matching techniques (e.g. COMA [DR02], COMA++[ADMR05], YAM++[NB12]).

3.2.3. Measuring Alignment Quality

Nonetheless, evaluation of a matching system is typically performed empirically. This requires (often manually) creating a reference alignment $\mathbb{A}_{\text{ref}}$ and comparing the alignment produced by the ontology matcher(s) against this reference alignment.

This comparison typically done using standard precision (def. 3.21) and recall (def. 3.22) measurements.

Definition 3.21 Alignment Precision

Let $\mathbb{A}$ be an alignment and $\mathbb{A}_{\text{ref}}$ a reference alignment. Then

$$\mathsf{Prec}(\mathbb{A}, \mathbb{A}_{\text{ref}}) \equiv_{\text{def}} \frac{|(\mathbb{A} \cap \mathbb{A}_{\text{ref}})|}{|\mathbb{A}|}$$

is the **precision** of $\mathbb{A}$ relative to $\mathbb{A}_{\text{ref}}$.

Definition 3.22 Alignment Recall

Let $\mathbb{A}$ be an alignment and $\mathbb{A}_{\text{ref}}$ a reference alignment. Then

$$\mathsf{Rec}(\mathbb{A}, \mathbb{A}_{\text{ref}}) \equiv_{\text{def}} \frac{|(\mathbb{A} \cap \mathbb{A}_{\text{ref}})|}{|\mathbb{A}_{\text{ref}}|}$$

is the **recall** of $\mathbb{A}$ relative to $\mathbb{A}_{\text{ref}}$

These basic scores are often combined into a single value, the *accuracy*, typically in the form of the $F1$-score or f-measure.

Definition 3.23 Alignment f-score

Let $\mathbb{A}$ be an alignment and $\mathbb{A}_{\text{ref}}$ a reference alignment. Then

$$\mathsf{F}(\mathbb{A}, \mathbb{A}_{\text{ref}}) \equiv_{\text{def}} 2 \times \frac{\mathsf{Prec}(\mathbb{A}, \mathbb{A}_{\text{ref}}) \times \mathsf{Rec}(\mathbb{A}, \mathbb{A}_{\text{ref}})}{\mathsf{Prec}(\mathbb{A}, \mathbb{A}_{\text{ref}}) + \mathsf{Rec}(\mathbb{A}, \mathbb{A}_{\text{ref}})}$$

is the f**-score** or **accurancy** of $\mathbb{A}$ relative to $\mathbb{A}_{\text{ref}}$.

All of these simple measurements, however, are unsatisfactory when the semantics of ontological models are considered. Euzenat and Valtchev [EV03] established in 2003 that the "classical" evaluation measures of precision and recall are of limited usefulness to measure the quality of an ontology alignment. Ehrig and Euzenat [EE05] later continue to venture in the same direction. A suitable measurement for the quality of an ontology alignment needs some notion of the *degree of correctness*, i.e. it should take into the account the distance between derived alignment correspondences and the reference alignment.

Semantic Precision and Recall Ehrig and Euzenat [EE05] propose a "relaxed" version of precision and recall that takes into account the different degrees of mismatch between alignments. Their version of such a measurement is presented in a later paper [Euz07] in terms of an *ideal* semantic precision and recall based on model theoretic observations.

Definition 3.24 α-Consequence

Given two ontologies O_1, O_2, and an alignment $\mathbb{A}$ (expressed as a set of correspondences $a = (e, e', r), a \in \mathbb{A}$), a correspondence δ is an **α-correspondence** of $(O_1, O_2, \mathbb{A})$, iff all models of δ also satisfy any of the correspondences $a \in A$.
The set of all α-correspondences of an alignment $\mathbb{A}$ for ontologies O_1, O_2 is called $\mathrm{Cn}(O_1, O_2, \mathbb{A})$ or short $\mathrm{Cn}(\mathbb{A})$ if O_1 and O_2 are clear from the context.

Consequently, the ideal precision and recall values of an alignment $\mathbb{A}$ with regard to a reference alignment $\mathbb{A}_{\mathrm{ref}}$ and two ontologies O_1, O_2 are given by definition 3.25 and 3.26.

Definition 3.25 Ideal Alignment Precision

Let $\mathbb{A}$ be an alignment and $\mathbb{A}_{\mathrm{ref}}$ a reference alignment. Then

$$\underset{\text{ideal}}{\mathsf{Prec}}(\mathbb{A}, \mathbb{A}_{\mathrm{ref}}) \equiv_{\mathrm{def}} \frac{|(\mathsf{Cn}(\mathbb{A}) \cap \mathsf{Cn}(\mathbb{A}_{\mathrm{ref}}))|}{|\mathsf{Cn}(\mathbb{A})|}$$

is the **ideal precision** of $\mathbb{A}$ relative to $\mathbb{A}_{\mathrm{ref}}$.

Definition 3.26 Ideal Alignment Recall

Let $\mathbb{A}$ be an alignment and $\mathbb{A}_{\text{ref}}$ a reference alignment. Then

$$\underset{\text{ideal}}{\mathsf{Rec}}(\mathbb{A}, \mathbb{A}) \equiv_{\text{def}} \frac{|(\mathsf{Cn}(\mathbb{A}) \cap \mathsf{Cn}(\mathbb{A}_{\text{ref}}))|}{|\mathsf{Cn}(\mathbb{A}_{\text{ref}})|}$$

is the **ideal recall** of $\mathbb{A}$ relative to $\mathbb{A}_{\text{ref}}$.

These measurements are semantically ideal, because they consider the actual semantic extension of the alignments, not their particular representation. Unfortunately, they also have a major drawback for practical use: they cannot be calculated in an efficient manner. When $\mathbb{A}_{\text{ref}}$ is a level 2 alignment, calculating $\mathsf{Prec}_{\text{ideal}}$ is most likely undecidable as there are infinitely many possible alignments.

These results seem discouraging, but they still serve to highlight an important point: semantics play a major role not only in the derivation of an alignment, but also in the evaluation of its correctness. Traditional measurements only give a superficial view of the quality of an alignment and careful evaluation is necessary to gain more insight on why a matcher might seem to perform poorly.

3.2.4. Current Developments and Challenges in Ontology Alignment

We have seen in the previous section (section 3.2.2), that a significant amount of research has been carried out to develop techniques for ontology matching. Results from the OAEI evaluation campaigns (e.g. [EMS+11]) also show that current matcher technology is well developed. Results of automated matching are not perfect, but results are more than sufficient to provide assistance during manual alignment between two ontologies.

On the infrastructure side, a standard method of representing alignments has been established in 2004 [Euz04]. The "Semantic Evaluation at a Large Scale" (SEALS) platform [WGCN12] provides a repository of test datasets and storage of evaluation results. In addition, SEALS defines a standard "bridge interface" in the Java programming language to be implemented by ontology matchers to support automatic evaluation.

Table 3.2. Number of Reference Alignments for the OAEI "conference" Dataset.

cmt-confOf	16	conference-confOf	15
cmt-conference	15	conference-edas	17
cmt-edas	13	conference-ekaw	25
cmt-ekaw	11	conference-iasted	14
cmt-iasted	4	conference-sigkdd	15
cmt-sigkdd	12	edas-ekaw	23
confOf-edas	19	edas-iasted	19
confOf-ekaw	20	edas-sigkdd	15
confOf-iasted	9	ekaw-iasted	10
confOf-sigkdd	7	ekaw-sigkdd	11
iasted-sigkdd	15		

Extracted from downloaded dataset, retrieved 2013-04-03
http://oaei.ontologymatching.org/2012/conference/index.html

However, ontology alignment is still a very active research area with many open challenges. This section gives an overview over current developments in ontology alignment and also gives pointers to unexplored areas in the domain. (as described by e.g. [SPM08, SE08, SJ13])

Large Datasets and Matcher Scalability

Evaluation of ontology matchers is largely done empirically. Creating reference datasets is a time-consuming task and hence existing reference alignments are often rather small. For example, the number of reference alignments for the "conference" dataset of the OAEI are listed in table 3.2, containing a maximum number of at most 25 (conference-ekaw) alignments.

Concerns have therefore been expressed

1. on the scalability of matcher technology to large datasets, and

2. on how to obtain pre-matched large datasets with minimal human effort for evaluation.

Recent research has placed a focus on the scalability of matcher technology. Approaches range from partitioning of ontologies into independently matchable sets (e.g. Falcon-AO [JHCQ05], LogMap [JRGZH12]) and re-use of already established alignments to the integration of upper-level ontologies to speed up the matching process (e.g. GOMMA [HGKR12]).

As an answer to the growing concerns of the applicability of existing matcher technology to large and real-world datasets, the benchmark sets for the OAEI campaign have been extended to also contain larger reference alignments. In particular, the "Large BioMed track" was established for OAEI 2011.5 [MSZT+12]. These alignment tracks compose large biomedical ontologies with $78,989$, and $66,7224$ classes, respectively. The reference alignment contains $2,989$ (simple) correspondences.

Results from the second 2011[9] (OAEI 2011.5 [MSZT+12]) and the 2012 evaluation campaign have shown that many –then current matching– systems were unable to handle such large matching tasks. Many newer matching systems, like LogMap [JRGZH12] and GOMMA [HGKR12], however, have shown that scalable matching even of large ontologies need not necessarily also yield a decrease in matching accuracy.

Use of Semantic Information

Many early matching systems took a very limited view on ontologies. Descended from graph and (relational) schema matching, the internal semantics of the ontological models were often ignored or only incorporated in the form of additional graph connections without really honoring their formal semantics. These matching systems have focussed on the linked-data and frame-based views on ontologies, neglecting the language based view.

In the recent years, more and more matching systems make use of the expressed formal semantics of the ontologies. Semantic information is used in two different contexts:

[9]There were two evaluation campaigns in 2011.

- to guide the matching process, and
- to verify alignment consistency and coherence.

Semantic Information To Guide the Matcher Some more modern matching systems are able to use semantic information to guide the matching process itself. For example, an iterative matcher could use consequences from already established correspondences in earlier steps to improve accuracy of correspondences derived in later steps. CODI [NNS11] uses topological similarity [Res99] to improve similarity computation in the concept hierarchy. LogMap [JRGZH12] extends the basic topological tree with information about disjoint classes.

Alignment Consistency and Coherence Transferring information from one ontology to the other does not guarantee the consistency of the information in the target ontology, automatically. As a simple example, a direct data property correspondence only copies data values from O_s to O_t. But if the constraint system OCS_t of O_t disallows certain values for the mapped data property that are allowed in the source ontology O_s, the mapped knowledge base will be inconsistent.

The situation is more complex, when there are multiple correspondences, whose consequences potentially interact. For example, when a matcher derives the correspondences $\beta_1 = (A, B, =, 1.0)$ and $\beta_2 = (A, C, =, 1.0)$ and the target OCS_t contains the assertion that $B \not\equiv C$, only one of the correspondences can be correct, because they contradict each other when applied together.

Inconsistency and incoherency can be grouped into different categories, depending on which type of interaction is causing the logical contradictions and when the problem occurs.

The first possible distinction is, if an inconsistency is *always* generated when considering one or a set of correspondences. Some of the logical contradictions appear independently of the knowledge bases mapped and some only appear when combined with a particular source knowledge base. Typically, only logical contra-

dictions that appear at the terminological level are considered, because it is almost always possible to purpose-construct knowledge bases that cause inconsistencies between alignment correspondences.

The simplest case is, when a correspondence is completely nonsensical in itself. This includes alignments that map a concept to $\bot$ or –in the case of bidirectional alignments– establish a relationship between two concepts already asserted to be disjoint.

Definition 3.27 Inconsistent Correspondence

A bridge rule β between two TBoxes TBox_1, TBox_2 is **inconsistent** iff

$$\mathsf{TBox}_1 \cup \mathsf{TBox}_2 \cup \{\beta\} \models \bot$$

for all knowledge bases KB_1 and KB_2 such that $\mathsf{TBox}_1 \cup KB_1 \not\models \top$ and $\mathsf{TBox}_2 \cup KB_2 \not\models \top$.
A correspondence (e_1, e_2, r, c) is **inconsistent with regard to source and target knowledge bases KB_1 and KB_2, respectively**, iff

$$\mathsf{TBox}_1 \cup KB_1 \cup \mathsf{TBox}_2 \cup KB_2 \models \top$$

but

$$\mathsf{TBox}_1 \cup KB_1 \cup \mathsf{TBox}_2 \cup KB_2 \cup \{\beta\} \models \bot.$$

Correspondence-level inconsistencies such as in definition 3.27 are usually quite rare. For two ontologies that do not share any concept or role references, it is not even possible to generate an inconsistent mapping that does not map directly to $\bot$. Many ontology matching systems, however, can generate sets of matching rules that cannot be applied together. A set of correspondences that contain interactions between rules that create inconsistencies when applied together is called *incoherent*.

Research in the direction of avoiding or repairing consistency and coherence problems has begun only quite recently. Systems that implement approaches to the consistency and coherence problem can be grouped into two different categories:

Definition 3.28 Incoherent Alignment

An alignment $\mathbb{A}$ with bridge rules $\beta_0, \beta_1, ...$ is **incoherent** with regard to a pair of TBoxes TBox_1, TBox_2 iff

$$\mathsf{TBox}_1 \cup \mathsf{TBox}_2 \cup \{\beta_0, \beta_1, ...\} \models \bot$$

for all knowledge bases KB_1 and KB_2 such that $\mathsf{TBox}_1 \cup KB_1 \not\models \top$ and $\mathsf{TBox}_2 \cup KB_2 \not\models \top$.
The alignment $\mathbb{A}$ is **incoherent with regard to knowledge bases KB_1 and KB_2**, iff

$$\mathsf{TBox}_1 \cup KB_1 \cup \mathsf{TBox}_2 \cup KB_2 \models \top$$

but

$$\mathsf{TBox}_1 \cup KB_1 \cup \mathsf{TBox}_2 \cup KB_2 \cup \{\beta_0, \beta_1, ...\} \models \bot.$$

those that implement consistency/coherence checking as a post-processing step after generation of alignments and systems that integrate the consistency checking procedure into the matching process.

A relatively simple post-processing "alignment repair" system has been demonstrated by Haase and Stojanovic [HS05a] in the context of ontology evolution. In their approach, they modify the ontologies themselves to find the largest ontology that is still consistent after new axioms were added to the ontology. A similar approach was proposed for ontology matching itself by Meilicke, Stuckenschmidt and Tamilin [MST07]. Their system considers alignments as bridge rules from distributed description logics [ST05]. The smallest set of conflicting bridge rules is searched and the correspondence e_1, e_2, r, c with the smallest confidence value c is repeatedly removed until consistency is regained. The system has been further developed into the ALCOMO [Mei11] system for repairing inconsistent ontology alignments. ASMOV [JMSK09] also implements a similar repair procedure by removing the mapping rule with the lowest confidence value when an inconsistency is detected. Two very successful (according to the OAEI campaign results [AGE+12]) matchers, CODI [NNS11] and LogMap [JRG11] implement elimination of inconsistent correspondences as part of the matching process.

Matcher Selection, Combination, and Parameter Tuning

Matcher Selection As noted in section 3.2.3, matcher performance is dependent on the nature of the involved ontologies. A lexical matcher without some form of multi-lingual thesaurus has trouble when two ontologies are formulated in different languages. A purely structural matcher has a hard time matching two ontologies with different layouts even if the terminology between both ontologies is re-used. Matchers that utilize external resources (e.g. using web search) are often unsuited for highly focussed ontologies.

Matcher Combination Progress has been made with regard to high-level combination of matcher results (COMA [DR02], COMA++[ADMR05]). Recents results from current ontology matchers show that purpose-designed methods to combine selected matching methods can yield improved results (e.g. YAM++[NB12]).

Parameter Tuning Another important aspect of optimizing the alignment process affects matchers that can be parameterized. Choosing a suitable set of parameters for a matcher usually requires in-depth knowledge of the matching system. Again, parameters suited for one alignment task can be unsuited for another alignment task. There is only limited initial research (e.g. [RP11]) available on automated parameter tuning for ontology matchers.

User Involvement

Regardless of the achievements of fully automated alignment systems, every generated aligment requires at least some corrective activity by the user. Interactive user interfaces to visualize correspondences and to repair generated alignments are typically quite limited. Those that exist (e.g. COMA++[ADMR05], Snoggle [RDB+08], AgreementMaker [CAS09], YAM++ [NB12]) usually provide a left-to-right view with the correspondences visualized as arrows between ontology elements. Support for interactive, iterative generation of alignments has only appeared quite recently (e.g. [NB12]).

Explaining With regard to consistency checking of formal knowledge models, large steps have been made by the research community with regard to generating *explanations* for inconsistencies. Explanations provide the ontology designer with a way to locate the possible mistakes in the ontology design.

Similar functionality would also be desirable for ontology alignment. Some matching systems base their decisions on prior information. For example, they use upper ontologies, generalize from existing alignments or incorporate user decisions into the matching process. Such systems greatly benefit from a method to show the effects of existing pieces of information.

Complex Alignments

Last but not least, there is the topic of generating complex alignments. With the exception of a few approaches (e.g. [RMŠZS09, Sch09, vZSS09, RVMŠZ10]), complex alignments are largely neglected by the research community. Research focus currently remains on matcher scalability and use of semantic information.

With regard to complex alignments, significant formalization work has been performed by Scharffe [Sch09] classifying a large amount of possible transformation patterns for ontology matching. As a practical result, a "pattern server" [vZSS09] has been implemented, that returns possible transformation patterns for an input ontology. At around the same time, linguistic analysis [RMŠZS09] has been successfully implemented for deriving certain complex alignments. Research in this direction, however, seems to have pattered out, with no new alignment systems supporting complex correspondences published since.

3.3 Document Ontologies

The ontology itself is a highly general knowledge representation framework with a wide range of application domains. Not every technique for ontology matching is suited for every type of ontology. For example, the OAEI evaluation datasets are divided into different challenges and many ontology matchers excel only in certain parts of the competition.

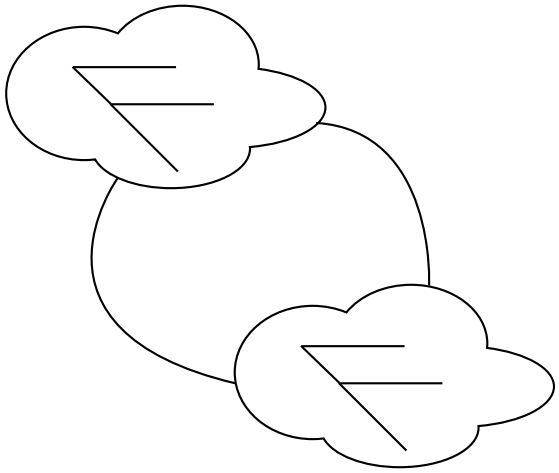
The literature contains little information on the suitability of matchers for certain ontology types. Most matchers only document the limitations of the types of ontologies they are able to process, but this restriction is usually at a syntactic level (e.g. if the matcher cannot handle transitive roles). Usually such restrictions are only mentioned in passing and the focus is on the description of the matcher algorithm and its evaluation. Suitability evaluations are mostly empirical, hence we mostly have information about particular cases where matcher m_1 is better suited to match ontology O_1 with ontology O_2 than matcher m_2. A systematic description of matcher suitability is missing from the literature, a fact probably owned to the complexity of the task in general.

In practice, this means that suitability of a matching system for a particular class of ontologies needs to be evaluated empirically. But even if pre-aligned reference datasets are available, the transferability of the evaluation results into a broader range of ontologies from a single domain seems dubious. There simply exists –as of this writing– no system to classify the suitability of a particular set of ontological models with regard to a particular type of matcher.

This section attempts an analysis of the type of "*document ontologies*" that are the primary target of the techniques developed in this thesis. The restrictions described here are intended to be from a modelling perspective, providing an overview over the overall structure of the expected ontologies. Syntactical restrictions of the individual algorithms (e.g. no qualified number restrictions) are discussed at a later point. I document the assumptions made about "*matchable*" ontologies and also list the known problems that the present work is not attempting to solve.

Our first observation is that the elements (concepts and roles) of document ontologies can be separated into two different categories: *primary* and *secondary*. The main difference between both is that instances of primary elements are largely independent and –usually– can exist by themselves within a knowledge base. Instances of secondary objects, on the other hand, can only exist when linked (transitively) to at least one primary element.

Figure 3.3. Structure of a Document Ontology



Digital maps described earlier are a prime example: the spatial entities form the set of the primary elements, while any "*attached*" information is provided mainly by secondary elements. As such, in the most simple case, every document ontology is a forest of linked individuals with the root nodes of the forest being instances of primary objects with no shared elements between (primary) trees.

In practice, however, such a strict design would be both too redundant and restrictive. For example, a separate "Material" class would be needed for every primary tree that has a *material-like* concept. Such a design –while common for some ontologies– looses the explicit semantic link between the different material types (i.e. all of them being materials). Instead, secondary elements will be most likely re-used between primary trees. The simple design is also too restrictive, because it disallows links between trees, thus severely limiting the representative power of the ontology. Instead, we can expect links to be present between the different trees, resulting in ontology schemas following the pattern as presented in fig. 3.3.

It is important to note that the resulting meta-structure is not restrictive. In a degenerate case, all concepts are primary. Connections are then also only between primary elements, which makes all connections also primary elements. The built heritage documentation ontologies on the other hand, have an almost pure forest structure. Most of the ontologies do not feature any links between primary trees.

And even though this is about to change when the potential of linked semantic data are being put to use in evolved ontologies, the forest structure is still very prominent.

It is possible to sketch an ontology alignment process that makes use of such a tree structure:

1. Obtain correspondences for the primary elements.
2. For each matched pair of primary elements (in different ontologies), unfold the primary tree. This results in a set of ontology fragments (partially with individuals). Find alignments between primary trees.
3. For each link between an pair of primary trees that is not a link between root nodes, find a suitable matching link and attachment points between the pair of aligned primary trees.

The approach tries to balance the problem of relative semantics of elements against computational feasibility of matching. The more context specific the interpretation of an element is, the more work is required from a matcher. Simple matchers usually have no (pure element-level matchers) or a limited view of the "surroundings" of an element. Their performance is best when the ontology consists of mostly primary elements. Making use of the tree structure, the primary trees become matchable elements by themselves without having to resolve to a full document-to-document match.

Step 1 can be solved by existing "simple" ontology matchers. This thesis will focus on expansion and tree-to-tree matching, making use of semantic techniques to expand and align primary trees. It is assumed that step 3 can be handled using appropriate post-processing methods.

SECTION 4.0

Automated Reasoning in LillyTab

This chapter introduces knowledge representation logics and –in particular– description logics (DLs) as the semantic foundation of many ontology modeling frameworks (section 4.1). Semantics and algorithms for reasoning in description logics are described in section 4.2.

The second part of the chapter introduces description logic tableau reasoning as a sound and complete reasoning method for description logic knowledge bases.

The final part introduces the DL reasoner "LillyTab" that was implemented as part of this thesis, outlines its architectural details and gives rationale for its implementation.

4.1 Knowledge Representation Logics

In section 3.1 the ontology has been introduced as a general model for knowledge representation based on the three basic elements: individuals, concepts (as sets of individuals), and relations (links between individuals). The basic model, however, has limited support for automated derivation of new knowledge from existing assertions. For example, if we wanted to model a university's employees and classify project staff, we would like to specify the constraint (in FOL):

$$\forall\, x, y\,.\, \mathsf{Staff}(x) \wedge \mathsf{worksOn}(x, y) \wedge \mathsf{Project}(y) \Rightarrow \mathsf{ProjectStaff}(x) \tag{4.1}$$

This is not supported by the basic ontological model we have considered so far. While the ontology frame (definition 3.4) allows construction of a subsumption hierarchy of named concepts, the above example can only be modelled by explicitly

asserting an employee to belong to ProjectStaff. If we want to model the inference in equation (4.1) ontologically, a formal system of inference is needed. This brings the focus back to the third view on ontologies, the language-based view (section 3.1.3), because formal systems of inference are almost exclusively specified as logical languages.

Such a logical language must not only be able to handle the assertions that we make about individuals (for example, in equation (4.1), that some employee is on staff), but it must also be able to model the constraints and inferences that we want to impose upon our ontological model (again like the constraint in equation (4.1)). And it also must support the desired automatic inference.

4.1.1. Automated Inference and Reasoning

Beyond the basic descriptive capabilities, the first requirement to formulate to a *formal knowledge system* is therefore, what kind of inferences it needs to perform, i.e. which services should be offered by the deduction system. Over time, a small set of *standard reasoning services* has emerged, which are expected from any modern knowledge representation system, if it offers automated reasoning:

satisfiability The check for satisfiability is performed to determine if there are no contradictions within a knowledge base definition. For example, if we repeat the constraint from equation (4.1) and (incorrectly) add the assertion, that Staff does not work on projects (i.e. that Staff and ProjectStaff are disjoint), ProjectStaff will be unsatisfiable.

$$\forall\, x\,.\,\exists\, y\,.\,\mathsf{Staff}(x) \wedge \mathsf{worksOn}(x,y) \wedge \mathsf{Project}(y) \Rightarrow \mathsf{ProjectStaff}(x)$$
$$\forall\, x,y\,.\,\mathsf{Staff} \wedge \mathsf{worksOn}(x,y) \Rightarrow \neg\mathsf{Project}(y)$$

Having defined a concept that can never be satisfied does not really make sense, so this is not allowed in most knowledge modelling frameworks.

subsumption Subsumption is the test if one set of individuals (defined by a set of constraints) is contained within another set. For example, if we wanted to know, if all ProjectStaff can(!) only work on Projects, we need to perform a subsumption test.

In logical languages, that support the negation of arbitrary concepts, subsumption can be reduced to satisfiability. Instead of testing if $\mathsf{Sub} \subseteq \mathsf{Sup}$, we can test, if $\mathsf{Sub} \cap \neg\mathsf{Sup}$ is satisfiable.

consistency Consistency checking (or instance testing) is performed on a knowledge base to verify that the asserted information in the knowledge base is consistent with the formulated constraints. A knowledge base is *consistent*, if its interpretation function ι fits with the constraints our logical language imposes upon the contained assertions.

For example, if we assert both that someone is not a member of ProjectStaff ($\neg\mathsf{ProjectStaff}(x)$) and at the same time demand, that he does indeed work on a project ($\mathsf{worksOn}(x, y) \wedge \mathsf{Project}(y)$), we know that this is inconsistent with our definition of ProjectStaff in equation (4.1). Consistency checking determines such inconsistencies.

coherence An ontology is incoherent, if all of its concept descriptions are forced to be empty, i.e. there cannot be an instance of any of the concepts in the ontology. Note that this does not mean that empty concepts are not acceptable in general. For example $A = \emptyset$ does not make an ontology incoherent by itself; individual empty classes are allowed. However, when all declared concepts in an ontology are empty, the ontology is *incoherent*

Algorithms to check for coherence and consistency are closely related but not fully equivalent.

classification The named concepts in an ontology usually form a terminological structure, the *subsumption hierachy*. This hierarchical structure is, however, usually not made fully explicit. Classification involves making the hierarchical structure explicit, i.e. finding all sub- and super-classes of named concepts.

> Classification can always be reduced to repeated subsumption testing.

There was an early enthusiasm in artificial intelligence about new "intelligent" systems that could perform logical deductions automatically. A very real problem with such systems was more or less ignored by the first systems of knowledge representation: automated reasoning in very expressive logics like FOL or even higher order logics [Lei94] is not only computationally intractable (NP-complete and worse), but FOL itself is already semi-undecidable. A reasoning system that implements reasoning for FOL always suffers from the very real risk, that a consistency (or satisfiability, or subsumption) checking algorithm runs into an endless loop and never terminates.

Using a logical language where the termination of a reasoning task is not guaranteed, is clearly a highly undesirable situation for any knowledge representation system. Using a less expressive, but decidable logical language, however, may not be sufficient to model the knowledge structure of a particular problem domain. This trade-off between expressiveness and tractability constitutes a fundamental challenge of knowledge representation until today. The challenge became even more pronounced, when it was realized, that the relationship between decidability and expressiveness of representation was a lot more subtle than expected. For example, Brachman and Levesque note

> "*We illustrate how great care needs to be taken in the design of a representational facility, even when our intuitions about the language tell us that it is a simple one. As it turns out, even an apparently modest representation language can prove intractable*" [BL84, p. 1].

Consequently, finding decidable fragments of FOL has become a challenge for the logics research community that continues until today (see e.g. [HSG04]). In this *quest for expressive decidability*, one class of logical languages has received particular attention: the class of terminological representation systems, a term which is nowadays used synonymously with the class of logical languages that form the core of these systems: *description logics*.

4.1.2. Description Logics

The distinguishing feature of *description logics* [BHP08] is the fact that they represent special purpose languages explicitly designed to build *terminological representation systems*. All description logics contain functionality to combine individuals into a named set (a concept) and to define relationships between these individuals (roles). That is, all description logics contain exactly those facilities necessary to build ontologies, namely (primitive) **concepts**, **roles**, and **individuals** (see section 3.1.2).

This is already true for the first general terminological representation language that has seen widespread publicity. KL-ONE [BS85] is fundamentally about the grouping and classification of objects (according to their properties) into sets of objects called *concepts* and it also supports links between individual objects.

KL-ONE also introduced the important distinction between *primitive* and *defined* concepts. Primitive concepts are sets of individuals that are accumulated under a common name (the concept) but for which no further definition is required/given for the target domain. Consequently, objects belonging to primitive concepts need to be asserted to do so explicitly.

Defined concepts, on the other hand, are (and in KL-ONE must be) formulated purely as restrictions on the properties of individuals. The individuals belonging to a defined concept are resolved by automatic deduction on the knowledge base. For example, KL-ONE can represent the ProjectStaff concept as follows[10].

$$\begin{aligned} &\mathsf{ProjectStaff} := \\ &\qquad \mathbf{AND}\ \mathsf{Staff}\ (\mathbf{SOME}\ \mathsf{worksOn}\ \mathsf{Project}) \end{aligned} \tag{4.2}$$

[10]The syntax used here is that of frame logic ($\mathcal{FL}$, see figure 4.1a), because no formal syntax for KL-ONE is given in the initial system description. $\mathcal{FL}$ appears shortly afterwards in a paper by the same authors [LB87]

Figure 4.1. Syntax of Frame Logic

$$
\begin{aligned}
C ::= & atom \\
& |(\text{AND } C \ldots C) \\
& |(\text{ALL } r\ C) \\
& |(\text{SOME } r)
\end{aligned}
\qquad
r ::= atom
$$

(a) $\mathcal{FL}^-$

$$
\begin{aligned}
C ::= & atom \\
& |(\text{AND } C \ldots C) \\
& |(\text{ALL } r\ C) \\
& |(\text{SOME } r)
\end{aligned}
\qquad
\begin{aligned}
r ::= & atom \\
& |(\text{RESTR } r\ C)
\end{aligned}
$$

(b) $\mathcal{FL}$

Because ProjectStaff is defined here in terms of its properties, it is a defined concept. KL-ONE's inference system was able to automatically classify individuals as ProjectStaff by considering their properties. In this case, any individual belonging to ProjectStaff needs also to belong to Staff and need be linked with an worksOn role to at least one (**SOME**) object that was classified as a Project.

It is this automated deduction system that was the novel part of KL-ONE, but that is also its primary drawback. KL-ONE suffered from the same problem that also plagues knowledge representation with full first order logic. Automated reasoning within such expressive systems is often undecidable (e.g. subsumption in KL-ONE is undecidable [SS88]), severely limiting their usefulness. Even more on the problematic side, it is not immediately discernible, which syntax "features" can be enabled for a logical language without any problems and which features (and interaction of features) cause (NP-)hardness or even undecidability. For example, in [BL84], the authors note that subsumption in even the frame language $\mathcal{FL}$ (figure 4.1a) is NP-complete, while nearly the same language $\mathcal{FL}^-$ (figure 4.1b), which is $\mathcal{FL}$ without the role restriction constructor, features polynomial time subsumption testing.

Based on these initial findings, diverse research has taken place to develop logical languages that provide both the necessary expressiveness but were still candidates for tractable automated reasoning. This research, however, held fast to the princi-

Figure 4.2. Syntax of the Description Logic $\mathcal{AL}$

The logic $\mathcal{AL}$ is defined by the language $L_{\mathcal{AL}}(\mathcal{I}, \mathcal{C}, \mathcal{R})$, which is the smallest set (i.e. with least fixed point semantics) that can be constructed from the following grammar from the starting symbol S using
the **atomic concept constructors**

$S ::= \top$

$S ::= \bot$

$S ::= C$, for all atomic concepts $C \in \mathcal{C}$

and the **non-atomic concept constructors**

$S ::= S \sqcap S$

$S ::= \neg C$, for all atomic concepts $C \in \mathcal{C}$

$S ::= \forall\, r\,.\, S$ for all $r \in \mathcal{R}$

$S ::= \exists\, r\,.\, \top$ for all $r \in \mathcal{R}$

where $\mathcal{R}$ is a set of of role names, and $\mathcal{C}$ is a set of named class identifiers.

ples that are already the building blocks of KL-ONE. The result of this research are the modern *description logics*, which are at the core of most state of the art ontology formalisms.

There is a large variety of logical languages that are called *description logics* today. These can be traced to three "ancestor" logics: attribute logic ($\mathcal{AL}$), the already mentioned frame logic ($\mathcal{FL}$) and $\mathcal{EL}$. All of these description logics are defined syntactically, i.e. by the constructs allowed in the syntax of valid formulæ. For example, the syntax of $\mathcal{AL}$ given in figure 4.2.

It can be seen, that –apart from the use of mathematical operators instead of textual symbols– this syntax is very similar to that of KL-ONE. Equation (4.3) shows the prototypical definition of a BiologicalFather-concept in $\mathcal{AL}$:

$$\mathsf{BiologicalFather} := (\mathsf{Male} \sqcap \exists\, \mathsf{ancestorOf}\,.\,\top) \tag{4.3}$$

$$\forall\, \mathsf{ancestorOf}\,.\,\mathsf{Person} \tag{4.4}$$

A BiologicalFather is both (intersection $\sqcap$) a male (i.e. belongs to the concept Male) and is connected to at least one individual by an ancestorOf relation, i.e. a father is a male that is the ancestor of someone. Note, that the concept definition operator ($:=$) is not part of the logical language.

It is however noticeable, that $\mathcal{AL}$ does not support the complex existential restriction from KL-ONE. It is not possible to write $\exists$ ancestorOf . Person, because this is not supported in $\mathcal{AL}$'s syntax. Equation (4.4) shows a supporting axiom and asserts globally, that the ancestorOf can only point to objects of type Person. The same definitions are also possible in $\mathcal{FL}$ as negation was not required and only the overlapping syntax elements of $\mathcal{AL}$ and $\mathcal{FL}$ are used. $\mathcal{EL}$ is an even simpler logic allowing only concept intersection ($C \sqcap D$) and qualified existential restrictions ($\exists\, r\,.\,C$, with C an arbitrary $\mathcal{EL}$-concept).

It can be seen that the syntax of these basic description logics is very similar to that of first order logic. An important difference between description logics and and full first order logic is, however, that formulæ do not express truth values, but rather that any formula is itself a concept description. *True* in $\mathcal{AL}$ is denoted by the *all encompassing* concept $\top$ and *false* is denoted by the empty concept $\bot \equiv_{\text{def}} \neg\top$. In equation (4.5), above, $(\mathsf{Male} \sqcap \exists\, \mathsf{ancestorOf}\,.\,\mathsf{Person})$ is the set of all individuals that are both Male and have at least one ancestorOf successor in the set of Persons. Furthermore, $\mathcal{AL}$ is a *variable-less* logic. The arguments to the $\exists$- and $\forall$-quantors are not variables but role names. And finally, $\mathcal{AL}$ does not support functions or n-ary predicates. The expression of connections between individuals is restricted to (binary) roles.

While the basic logics are often sufficient to describe many real world scenarios (for example, the SNOMED RT [SCC$^+$97] ontology uses only $\mathcal{EL}^{++}$ an extension of $\mathcal{EL}$), their expressiveness is, nonetheless, somewhat unsatisfactory and incom-

plete. For example, if we wanted to create a Father-concept that does include biological and legal fathers, we cannot re-use the (potentially) existing concepts BiologicalFather and LegalFather:

$$\mathsf{Father} := \mathsf{Male} \sqcap ((\exists\, \mathsf{ancestorOf} \,.\, \top) \sqcup (\exists\, \mathsf{legalFatherOf} \,.\, \top)) \tag{4.5}$$

Concept union (and non-atomic negation) are missing from all the basic languages, which leaves all the logics somewhat *one-sided*, as not every concept description also has a negative counterpart. This problem was first remedied by Schmidt-Schauß and Smolka, who gave syntax, semantics, and complexity results for an extended version of attribute logic, dubbed *attribute logic with complements* ($\mathcal{ALC}$, [SSS91]). $\mathcal{ALC}$ (figure 4.3) forms the basic logic underlying all modern description logics as it is the first that is closed under concept negation.

Definition 4.1 Formal Semantics for $\mathcal{ALC}$

Given a set of named concepts $\mathcal{C}$, a set of roles $\mathcal{R}$, and a set of individuals $\mathcal{I}$, the interpretation of a formula $\phi \in \Sigma_{L_{\mathcal{ALC}}}$ is defined via an **interpretation function** $\iota \equiv_{\text{def}} \iota_{\mathcal{I}} \cup \iota_{\mathcal{R}} \cup \iota_{\mathcal{C}}$ with

- $\iota(\{\underline{a}\}) = \iota_{\mathcal{I}}(\{\underline{a}\}) = \underline{a}$ for all $\underline{a} \in \mathcal{I}$
- $\iota(C) = \iota_{\mathcal{C}}(C) \subseteq \mathcal{I}$
- $\iota(r) = \iota_{\mathcal{R}}(r) \subseteq \mathcal{I} \times \mathcal{I}$

such that for each subformula it holds recursively

- $\iota(C \sqcap D) = \iota(C) \cap \iota(D)$
- $\iota(C \sqcup D) = \iota(C) \cup \iota(D)$
- $\iota(\neg C) = \mathcal{I} - \iota(C)$
- $\iota(\forall\, r \,.\, C) = \{x \in \mathcal{I} | \forall\, y \in \mathcal{I} \,.\, (x, y) \in \iota(r) \Rightarrow y \in \iota(C)\}$
- $\iota(\exists\, r \,.\, C) = \{x \in \mathcal{I} | \exists\, y \in \mathcal{I} \,.\, (x, y) \in \iota(r) \wedge y \in \iota(C)\}$

Table 4.1. Interpretation-Preservation Transformations in $\mathcal{ALC}$

$\neg\neg C$	$\equiv C$	$A \Rightarrow B$	$\equiv \neg A \sqcup B$
$A \sqcup B$	$\equiv \neg(\neg A \sqcap \neg B)$	$A \sqcap B$	$\equiv \neg(\neg A \sqcup \neg B)$
$\forall r \,.\, A$	$\equiv \neg\exists r \,.\, \neg A$	$\exists r \,.\, A$	$\equiv \neg\forall r \,.\, \neg A$
$(A \sqcup B) \sqcap C$	$\equiv (A \sqcap C) \sqcup (B \sqcap C)$	$(A \sqcap B) \sqcup C$	$\equiv (A \sqcup C) \sqcap (B \sqcup C)$
$(A \equiv B)$	$\equiv (A \Rightarrow B) \sqcap (B \Rightarrow A)$		

Formal semantics for $\mathcal{ALC}$ are shown in definition 4.1. Within $\mathcal{ALC}$, the negation of every concept description is also a valid concept description in the same logic. Consequently, concept union is well defined by the negation of intersection. $\mathcal{ALC}$ also enables the "usual" syntactic transformations known from propositional and first order logic (table 4.1).

Because the logic is closed under negation, $\mathcal{ALC}$ formulæ can be converted into *negation normal form* (NNF).

Definition 4.2 Negation Normal Form

A formula ψ is in negation normal form iff for any negation $\neg A$ that appears in ψ, A is an atomic concept, i.e. $A \in \mathcal{C}$.

Most reasoners support rewriting a formulae into negation normal form or require that the input formulæ already are in negation normal form. Formulæ in NNF contain fewer logical axiom types and the individual axioms are less complex than what is permitted by the language in general. Making use of NNF thus simplifies reasoning algorithms considerably. Rewriting a formula to NNF is easily possible using the transformations from table 4.1.

Reasoning in $\mathcal{ALC}$ is decidable. Satisfiability testing and concept subsumption are, however, PSPACE-complete [SSS91]. $\mathcal{ALC}$ is therefore not a logical language that seems suitable for ontology modelling tasks, since reasoning effort still seems to grow exponentially with ontology size (at least with known algorithms and if $P \neq NP$). Fortunately, the worst case of exponential complexity seems to be quite rare for "typical" ontological models. Most ontologies rarely see (as noted above, the SNOMED RT ontology uses only [SCC+97] $\mathcal{EL}^{++}$) the constructs that cause

Figure 4.3. Syntax of the Description Logic $\mathcal{ALC}$

The logic $\mathcal{ALC}$ is defined by the language $L_{\mathcal{ALC}}(\mathcal{I}, \mathcal{C}, \mathcal{R})$, which is the smallest set (i.e. with least fixed point semantics) that can be constructed from the following grammar from the starting symbol S using the **atomic concept constructors**

$S ::= \top$

$S ::= \bot$

$S ::= C$, for all atomic concepts $C \in \mathcal{C}$

and the **non-atomic concept constructors**

$S ::= S \sqcap S$

$S ::= S \sqcup S$

$S ::= \neg S$

$S ::= \forall\, r\,.\, S$ for all $r \in \mathcal{R}$

$S ::= \exists\, r\,.\, S$ for all $r \in \mathcal{R}$

where $\mathcal{R}$ is a set of of role names, and $\mathcal{C}$ is a set of named class identifiers.

combinatorial explosion. As a result, it was possible to construct special purpose reasoning systems, that –despite of their worst case exponential running times– perform quite well in practice.

4.1.3. Expressive Description Logics

The encouraging results in dealing with theoretically intractable algorithms that nonetheless could be modified to yield satisfying results for all practical matters has lead to the development of many different extensions to the basic description logic $\mathcal{ALC}$. As a result, there is currently a "zoo" of description logic languages based on the original $\mathcal{ALC}$. These various languages have been categorized into a (semi-)formal naming system based on syntactic features available in the respective logic. This categorization systems makes mostly use of single letters or short expressions. An overview is also given in [KSH12].

$\mathcal{S}$ is $\mathcal{ALC}$ with support for *transitive roles*.

In $\mathcal{S}$ it is possible to flag any role as transitive. For each triple of individuals $\underline{a}, \underline{b}, \underline{c}$, this means that if $\underline{a}$ is connected to $\underline{b}$ and $\underline{b}$ is connected to $\underline{c}$, there is also a mandatory (inferred) connection from $\underline{a}$ to $\underline{c}$. Formally,

$$\mathsf{transitive}(r) \equiv_{\mathrm{def}} \forall x, y, z \,.\, (r(x, y) \wedge r(y, z)) \Rightarrow r(x, z) \tag{4.6}$$

Transitivity can be enabled on a role-by-role basis. Reasoning in $\mathcal{S}$ is also PSPACE-complete [BHP08].

$\mathcal{I}$ inverse properties. This allows for formulating the constraint that one property is always the inverse of another. Formally,

$$\mathsf{inverse}(r, p) \equiv_{\mathrm{def}} \forall x, y \,.\, (r(x, y) \Leftrightarrow p(y, x)) \tag{4.7}$$

$\mathcal{U}$ indicates support for *concept union*

This is included in $\mathcal{ALC}$, but can be used as an extension symbol for $\mathcal{EL}$ and $\mathcal{FL}$.

$\mathcal{E}$ indicates support for full existential qualification.

This is also included in $\mathcal{ALC}$ and $\mathcal{EL}$, but can be used as an extension symbol for $\mathcal{FL}$. Note that $\mathcal{ALC}$ and $\mathcal{ALUE}$ are equivalent, but the shorter abbreviation is much more common.

$\mathcal{F}$ indicates support for *functional roles*, i.e. roles where each individual can have at most one successor linked by the same functional role.

$$\mathsf{functional}(r) \equiv_{\text{def}} \forall x, y, z \,.\, r(x, y) \wedge r(x, z) \Rightarrow y = z \tag{4.8}$$

$\mathcal{H}$ indicates support for *role hierarchies*. Formally

$$r \sqsubseteq p \equiv_{\text{def}} \forall x, y \,.\, r(x, y) \Rightarrow p(x, y) \tag{4.9}$$

Reasoning in $\mathcal{SH}$ is EXPTIME-hard (hardness results from [Sch91], upper bound for the extension $\mathcal{SHIQ}$ proven in [Tob01]).

$\mathcal{R}$ indicates limited complex role inclusion, reflexive and irreflexive roles and role disjointness. The above mentioned $\mathcal{EL}^{++}$ is $\mathcal{ELRO}$.

$$\mathsf{reflexive}(r) \equiv_{\text{def}} \forall x \,.\, r(x, x) \tag{4.10}$$

$$\mathsf{irreflexive}(r) \equiv_{\text{def}} \forall x \,.\, \neg r(x, x) \tag{4.11}$$

$$\mathsf{disjoint}(r, p) \equiv_{\text{def}} \forall x, y \,.\, r(x, y) \Rightarrow \neg r(y, x) \tag{4.12}$$

$$r_1 \circ r_2 \sqsubseteq r_3 \equiv_{\text{def}} \forall x, y, z \,.\, r_1(x, y) \wedge r_2(y, z) \Rightarrow r_3(x, z) \tag{4.13}$$

Property chain inclusions (e.g. $r_1 \circ r_2 \sqsubseteq r_3$) must be cycle free.

For OWL2, $\mathcal{R}$ also indicates support for asymmetric roles and negative role assertions.

$$\mathsf{asymmetric}(r) \equiv_{\text{def}} \forall x, y \,.\, r(x, y) \Rightarrow \neg r(y, x) \tag{4.14}$$

A negative role assertion $\neg r(x, y)$ states that y is not an r-successor of x.

$\mathcal{N}$ indicates support for unqualified number restrictions. This allows to limit the number of role-successors of a particular individual above and below a fixed number.

$$\leq_n r \,.\, \top \equiv_{\text{def}} \{x | \# \{y | r(x, y) \in E\} \leq n\} \tag{4.15}$$

$$\geq_n r \,.\, \top \equiv_{\text{def}} \{x | \# \{y | r(x, y) \in E\} \geq n\} \tag{4.16}$$

Same complexity as $\mathcal{SH}$.

$\mathcal{Q}$ indicates support for qualified number restrictions. Qualified number restrictions allow arbitrary concept names (and not only $\top$) at the end of the number restriction. Includes $\mathcal{N}$.

$$\leq_n r \,.\, A \equiv_{\text{def}} \{x | \# \{y | r(x, y) \in E \wedge y \in A\} \leq n\} \tag{4.17}$$

$$\geq_n r \,.\, A \equiv_{\text{def}} \{x | \# \{y | r(x, y) \in E \wedge y \in A\} \geq n\} \tag{4.18}$$

Allowing this for transitive roles risks undecidability [HST00a]. If only simple roles are allowed, reasoning is EXPTIME-complete.

The web ontology language OWL [MvH04] is based on $\mathcal{SHOIN}(\mathbf{D})$, its subset OWL-Lite is $\mathcal{SHIF}(\mathbf{D})$. OWL 1.1 is $\mathcal{SHOIQ}(\mathbf{D})$

$\mathcal{O}$ support for *nominals*. Nominals are special concepts descriptors of the form $\{\underline{a}\}$ with $\underline{a} \in \mathcal{I}$, interpreted as singleton concept. $\{\underline{a}\}$ is closed and has only a single member $\underline{a}$. This makes it possible to create concepts that are enumerations of individuals. The nominal concept appeared first in KL-ONE, where it was named *nexus*.

The second version of the web ontology language OWL2 [MPSH08] is based on $\mathcal{SROIQ}(\mathbf{D})$.

(D) Support for datatypes. This both enables flagging of certain properties as *datatype properties* and the use of *data type ranges*. One can think of data type ranges as pre-defined concepts that contain only the individuals (then called *literals*), that are instances of the specified data type range.

The exact kind of datatype support implied by $(\mathbf{D})$ is not necessarily fixed and has changed over time. OWL1 [MvH04] has support for XML Schema datatypes [MB04]. OWL2 adds support for compound datatypes that can also assert restrictions across multiple property values. $(\mathbf{D})$ therefore only indicates any kind of datatype support.

Other syntactic additions include the self-restriction, that contains all individuals that are reflexively connected to themselves

$$\mathsf{self}(r) \equiv_{\mathrm{def}} \{x | r(x,x)\} \tag{4.19}$$

4.1.4. Description Logic Knowledge Bases

Knowledge bases in description logics are defined via logical axioms. The relevant axioms are usually split into distinct sets along the same lines as outlined in section 3.1. A description logics ontology consists of a TBox, an RBox, and an ABox.

The ABox contains only two kinds of assertions: class assertions and property assertions. Class assertions are of the type $C(\underline{a})$ and assert that the individual $\underline{a}$ belongs to the concept C. C usually is a named concept, it may however be a complex concept description. The second type of ABox assertions are property assertions. These are of the form $r(\underline{a}, \underline{b})$ and indicate that there is an r-link from $\underline{a}$ to $\underline{b}$.

While the ABox makes assertions about single individuals or pairs of individuals, global axioms are stored in the TBox of the ontology.

Definition 4.3 Description Logic ABox

A **description logic** ABox is a finite set of axioms of the form

- $C(\underline{a})$ with $C \in \Sigma_L$ and $\underline{a} \in \mathcal{I}$ an individual identifier.
  $C(\underline{a})$ asserts that the individual $\underline{a}$ belongs to the concept C.
- $r(\underline{a}, \underline{b})$ with $r \in \mathcal{R}$ and $\underline{a}, \underline{b} \in \mathcal{I}$ individual identifiers.
  $r(\underline{a}, \underline{b})$ asserts that there is an r-connection from $\underline{a}$ to $\underline{b}$.

Definition 4.4 Description Logic TBox

A **description logic** TBox is a finite set of axioms TBox $\subset \Sigma_L$.
The axioms in TBox hold globally for all individuals in a description logic ontology.
(L is the language of the respective description logic in use and Σ_L its signature.)

Assertions about the constraints imposed on properties (or roles) within the ontology are not considered part of an ontology's TBox, but part of an additional set called the RBox of the ontology. The RBox contains information about role hierarchy ($r_1 \sqsubseteq r_2$), role disjointness (disjoint(r, p)), transitivity (transitive(r)), and so on.

Definition 4.5 Description Logic RBox

A **description logic** RBox is a finite set of axioms specifying the relationship of roles within a description logic ontology. Let $r_1, r_2, r_3 \in \mathcal{R}$ be roles, RBox axioms are then of the form

- $r_1 \sqsubseteq r_2$ asserts that r_1 is a *subrole* of r_2,
- transitive(r) asserts role transitiveness,
- functional(r) asserts a functional role,
- inverse_functional(r) asserts an inverse functional role.,
- reflexive(r) asserts a reflexive role,
- disjoint(r, p) asserts role disjointness, and
- $r_1 \circ r_2 \sqsubseteq \neg r_3$

As with general ontologies (definition 3.6), description logic knowledge bases have a concept of closure and inference for role connections. $r\downarrow$ and $r\uparrow$ gain more expressed semantics:

Definition 4.6 DL Sub- and Superrole Closure

The relation $r\downarrow \equiv_{\text{def}} \{s | \forall\, x, y\,.\, s(x,y) \Rightarrow r(x,y)\}$ is called the **subrole closure of** r.
The relation $r\uparrow \equiv_{\text{def}} \{s | \forall\, x, y\,.\, r(x,y) \Rightarrow s(x,y)\}$ is called the **superrole closure of** r.

4.2 Automated Reasoning in Description Logics

The most important benefit of using description logics as the formal foundation of ontology modelling languages is of course the ability to perform the reasoning tasks as outlined in section 4.1.1. Reasoning in expressive DLs is usually performed as testing for satisfiability. As noted in section 4.1.1, subsumption testing can be reduced to satisfiability, if the underlying logic supports negation of arbitrary concepts. This is beneficial, because the same set of algorithms can be used for both reasoning tasks.

Two principal methods for satisfiability testing for DLs are in common use: resolution and model construction.

resolution Satisfiability testing via resolution works by transforming DL knowledge bases into a non-recursive logic program, preserving satisfiability [HV05, Mot06]. Satisfiability testing is then performed using known algorithms for deductive query answering (DataLog).

Resolution based calculi have some interesting properties especially with regard to large knowledge bases, because results from the theory on deductive databases can be re-used.

model construction By far the more common method for satisfiability testing for DL ontologies is based on the construction of representative pseudo models. A DL knowledge base is transformed into an initial DL pseudo-model and *completion rules* are iteratively applied to transform the initial model into a more refined one.

The principal decision procedure is also known as the "method of the analytic tableaux" (whereas tableau is singular and tableaux is plural). Tableau methods have been first developed for propositional logics and are available for many other formal logics [Smu95].

4.2.1. Basics of Tableaux Reasoning

Tableaux reasoning is by far the more common reasoning method. Most known DL reasoning systems (e.g. RACER [HM01], Pellet [SP04], Fact++ [TH06], Hermit [SMH08], TrOWL [TP10], ELK [KKS11]) are based on tableaux calculi and tableaux reasoners are usually the only ones that have support for the most expressive description logics.

The feature set of typical tableau reasoners, however, is somewhat varied[11]. Many "simple" reasoners support only less expressive logics (but are often very fast within their limited domain). Additionally, some reasoners only support pure TBox reasoning and cannot deal with ABox-assertions within their knowledge bases. Since existing documents will always contain ABox-assertions in their documents, we need a reasoner that supports reasoning with ABoxes.

Completion Graphs

A logical *tableau* consists of both a data structure (usually a graph) as well as a set of rules that modify the data structure. In the literature the term "tableau" is often used interchangeably for both the data structure as well as for the whole operation including the rule application.

Tableaux for description logics usually operate on a *completion graph*. A completion graph is a *pseudo-model* and thus a direct representation of a description logic knowledge base. The vertices of the completion graph represent individuals and the edges represent role connections between these individuals. The vertices are

[11] A list of DL reasoners and their capabilities is maintained at http://www.cs.man.ac.uk/~sattler/reasoners.html (last retrieved 2013-04-11 17:25 MEST).

labelled with concepts and the edges are labelled with role names. A formal tuple-based definition of the completion graph structure is given in definition 4.7. The notation used here is similar to that used in the $\mathcal{SHOQ}$-tableau [HS01], but uses an explicit set E for role assertions (instead of overloading $\mathcal{L}$ for both concept and role assertion).

Definition 4.7 DL Completion Graph

A **DL completion graph** over a logical language $L(\mathcal{I}, \mathcal{C}, \mathcal{R})$ (with signature $\Sigma_{L(\mathcal{I},\mathcal{C},\mathcal{R})}$) is a 3-tuple $G \equiv_{\text{def}} (V, E, \mathcal{L})$ with

V a set of individuals $V \subset N$ forming the vertices of a graph.
N is a totally ordered, countable, infinite set of node identifiers.

E a set of edges labelled with role names from $\mathcal{R}$, $E \subseteq \mathcal{R} \times V \times V$.
We write $r(x, y) \in E$, if $(r, x, y) \in E$.

$\mathcal{L}$ a set of concept labels. $\mathcal{L} \subseteq V \times \Sigma_{L(\mathcal{I},\mathcal{C},\mathcal{R})}$.
We write $(x : C) \in \mathcal{L}$ if $(x, C) \in \mathcal{L}$ and $\mathcal{L}[x] \equiv_{\text{def}} \{C | (x : C) \in \mathcal{L}\}$.

I will use the letter G to reference completion graphs. If more than one graph is present, graphs will be identified by index numbers in arabic numerals: G_0, G_1, ... An emboldened letter $\mathbb{G}$ will be used for sets of completion graphs, potentially with an index $\mathbb{G}_s$, $\mathbb{G}_t$, ... if there are multiple such sets, e.g. when considering mappings between source (s) and target (t) ontologies.

Since the completion graph is a representation of a DL knowledge base, a similar notion of *consistency* (definition 3.12) exists. It is possible to determine (often via a suitable algorithm) if a completion graph is consistent or inconsistent.

Definition 4.8 Consistency of a Completion Graph

Let $\mathbb{G}$ be the set of completion graphs over a description logic language $\Sigma_{L_{\text{DL}}}$. There exists a function

$$\text{consistent} : \mathbb{G} \mapsto \{\top, \bot\}$$

such that $\text{consistent}(G) = \top$ iff G is a model.
If $\text{consistent}(G)$, G is called **clash free**.
If $\neg\, \text{consistent}(G)$, G is said to **contain a clash**.

Since many of the algorithms in this thesis work by extending a completion graph, it is convenient to provide some shortcut definitions for often used idioms:

Inferability *Inferability* (definition 4.9) is the notion that some concept can be logically inferred from information contained inside the completion graph.

We write

$$G \models (x : C)$$

to assert that C is inferable in G at x. The semantics are probably best explained by contradiction: Inserting $\neg C$ into $\mathcal{L}[x]$ would make G inconsistent.

Definition 4.9 Concept Inferability in a Completion Graph

Let $G = (V, E, \mathcal{L})$ be a completion graph, $x \in V$ a node inside G, and $C \in \Sigma_{L_{\mathrm{DL}}}$ (with $\Sigma_{L_{\mathrm{DL}}}$ the concept language for $\mathcal{L}$).
C is **inferable** from G at x, iff $G' = (V \cup \{x\}, E, \mathcal{L} \cup \{\neg C\})$ is inconsistent
$G \models (x : C)$ is a shortcut for *C is inferable from G at x.*

Capacity The *capacity* (definition 4.10) of a completion graph is the set of concepts that can be added to a completion graph node without making it inconsistent.

There is a duality between the inferability and the capacity. If $(x : C)$ is in the capacity of G, then

$$G \not\models (x : \neg C)$$

Inserting a term from the capacity of G into G does not cause inconsistency.

Definition 4.10 Capacity of a Completion Graph

Let $G = (V, E, \mathcal{L})$ be a completion graph, $x \in V$ a node inside G, and $\Sigma_{L_{\mathrm{DL}}}$ the concept language for $\mathcal{L}$.
The **capacity** of G at x is the set

$$\left\{ C \in \Sigma_{L_{\mathrm{DL}}} \middle| \, \mathsf{consistent}((V \cup \{x\}, E, \mathcal{L} \cup \{(x : C)\})) \right\}$$

Semantic Extension Lattice The *capacity* forms the set of all consistent completion graphs into a partial order. This is the same partial order implied by the monotonicity of reasoning of the underlying description logic. Consequently, completion graphs can be put into a partial order of inferability.

A completion graph G_2 is a **semantic extension** of another completion graph G_1 (written as $G_1 \leq_{\text{ext}} G_2$) if all statements that can be derived in G_1 also hold in G_2 (but not necessarily the other way round).

Definition 4.11 Completion Graph Extension

Let $G_1 = (V_1, E_1, \mathcal{L}_1)$ and $G_2 = (V_2, E_2, \mathcal{L}_2)$ be completion graphs, and $\leq_{\text{ext}}$ be a $\mathbb{G} \times \mathbb{G} \mapsto \{\top, \bot\}$-relation such that

$$G_1 \leq_{\text{ext}} G_2 \Leftrightarrow \forall\, C \in \Sigma_{L_{\text{DL}}}, x \in (V_1 \cup V_2)\,.\,(G_1 \vDash (x : C)) \Rightarrow (G_2 \vDash (x : C))$$

If $G_1 \leq_{\text{ext}} G_2$, G_2 is called a **semantic extension** or simply **extension** of G_1.

Tableau Saturation

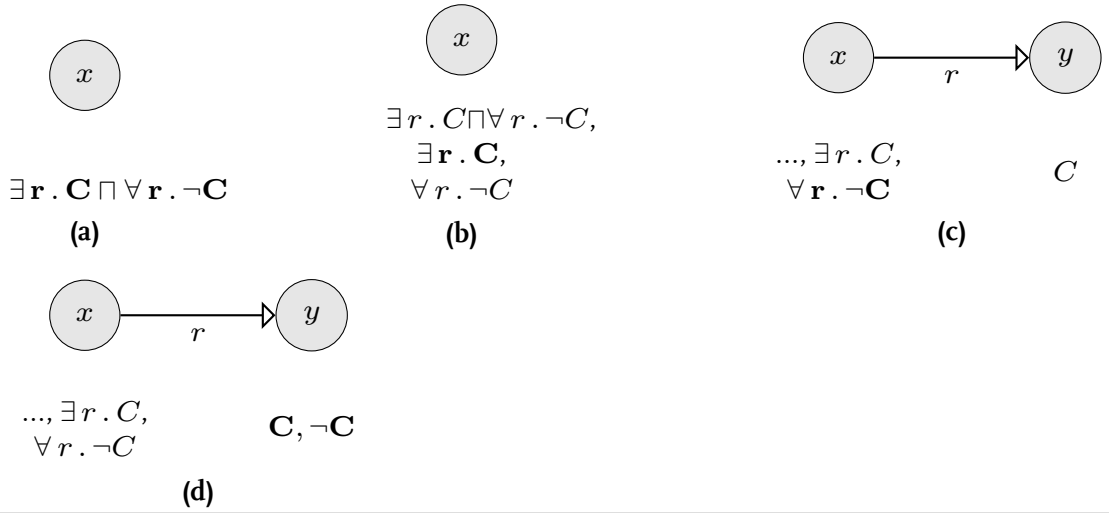
The abstract tableau process starts of with set a $\mathbb{G}_B$ of completion graphs to check for satisfiability. The individual graphs $G_b \in \mathbb{G}_B$ are called *branches* and $\mathbb{G}_B$ is sometimes called the *branch queue*.

When checking the satisfiability of a single concept C, $\mathbb{G}_B$ is usually a singleton set consting of the completion graph $G_0 = (V_0, E_0, \mathcal{L}_0)$ with $V_0 = \{x\}$, $E_0 = \emptyset$, $\mathcal{L}_0 = \{(x : C)\}$. If the tableau method finds at least one saturated graph by starting from G_0, we have found a model for C, hence C is satisfiable.

The rules of the tableau are applied to all of the graphs in $\mathbb{G}_B$, usually in order, but some reasoners also implement parallelization. Most of the time, the tableau modifies a completion graph in the branch queue in place. That is, it picks a completion graph from the branch queue, applies one or more transformations to the graph and then puts it back.

There are two exceptions to this rule:

Figure 4.4. Example Tableau Completion



1. When an inconsistency is found inside a completion graph, the respective graph is discarded and removed from the queue.

2. The tableau rules indicate that it is necessary to pursue two or more different paths. In this case, one or more copies (clones) of the current completion graph are made and modified independently. This process is known as *branching*. During branching, the size of the branch queue grows.

There are also two different termination conditions for the tableau process

1. When the tableau encounters a consistent completion graph and is unable to apply any more rules. In that case, the completion graph is called *saturated*.

   Reaching saturation indicates that the reasoning process has found a model of the initial graph, meaning that the initial graph is consistent. In that case, reasoning terminates with a successful result.

2. When every completion graph on the branch queue turns out to be inconsistent, the branch queue will eventually turn up empty.

   In this case, the initial completion graph was inconsistent and reasoning terminates unsuccessfully.

For example, consider the $\mathcal{ALC}$ concept description $\exists\, r\,.\, C \sqcap \forall\, r\,.\, \neg C$. It should be easy to see that this concept is not satisfiable, as the qualified concepts of the $\exists$ and $\forall$ terms are contradictory. To test for satisfiability of this concept description, a tableau reasoner creates an initial completion graph $G_0 = (V_0, E_0, \mathcal{L}_0)$ with a single individual labelled with the starting concept: $V_0 = \{x\}$, $E_0 = \emptyset$, $\mathcal{L}_0 = \{(x : \exists\, r\,.\, C \sqcap \forall\, r\,.\, \neg C)\}$ (figure 4.4a). Initially, the tableau evaluates the conjunction at x and splits it following the usual semantics of concept intersection: $(\exists\, r\,.\, C \sqcap \forall\, r\,.\, \neg C \Rightarrow (\exists\, r\,.\, C \wedge \forall\, r\,.\, \neg C))$ (figure 4.4b). The initial tableau rule has now transformed the initial conjunction into two smaller terms. This process is now repeated (with different transformation rules) with all remaining complex concepts.

The term $\exists\, r\,.\, C \in \mathcal{L}_0[x]$ demands that x has some arbitrary successor that is tagged with C. Since x does not already have such a successor, the tableau *invents* a new individual y and connects it to x, adding C to $\mathcal{L}_0[y]$ (figure 4.4c).

y is (initially) an *anonymous* node (definition 4.12), because it does not have a name in the shape of a nominal associated with it.

Definition 4.12 Anonymous Node in a Completion Graph

Let $G = (V, E, \mathcal{L})$ be a completion graph over a description logic with nominals. A node $x \in E$ is called **anonymous** iff $\nexists \underline{a}\,.\, \{\underline{a}\} \in \mathcal{L}[x]$. Otherwise it is called a **named** node.
Thus, a node is anonymous, if it does not have an associated nominal.

After the creation of y and the addition of $(y : C)$, we find that C is an atomic concept and no further transformation is possible. The remaining untransformed concept is $\forall\, r\,.\, \neg C \in \mathcal{L}_0[x]$. This demands that every r-successor of x is tagged with $\neg C$. Since x has one r-successor (namely y), $\neg C$ is propagated over the r-connection to y, leading to the final state of the completion graph depicted in figure 4.4d. In this final version, every complex concept has been transformed into one or more simple concepts. Most importantly, the contradiction embedded in the initial concept is readily apparent in the final completion graph. y needs to satisfy both C and $\neg C$, which is impossible. It is said $\mathcal{L}[y]$ contains a *clash*.

This simple example shows the basic operation of a logical tableau. In line with the observations at the start of section 4.1.1, an increase in expressiveness of the underlying logic, however, is usually combined with an disproportionally larger increase in tableau complexity. Modern description logics hover very close to the boundary of undecidability (e.g. [HKS06]). Tableaux for expressive description logics are very complex and need to employ heuristic expansion and other optimizations to be used in practice.

TBox and ABox Reasoning

Tableau algorithms can be separated into two categories: pure TBox reasoners can only check the satisfiability of a concept with regard to an empty ABox. This means, that TBox reasoners always start with a completion graph that has only a single node x_0 with $\mathcal{L}[x_0] = \{C\}$ and C the concept to test for satisfiability. For TBox reasoners (and if the description logic does not support nominals), the completion graph is also often a tree. When an arbitrary number of initial individuals (i.e. an ABox) is present, the nodes can be arbitrarily connected, forming a forest structure with multiple root nodes.

A common path is that –for a new logic– the TBox tableaux are constructed first, as they are easier to handle. The corresponding ABox tableau is often an extension of the TBox tableau. For example, the ABox tableau for $\mathcal{SHIQ}$ [HST00a] is based on the corresponding TBox tableau [HST00b] with some extensions. For description logics with support for nominals (the $\mathcal{O}$ family) the separation between TBox and ABox reasoning is lost. Because nominals can be used to introduce arbitrary individuals (i.e. new roots in the completion forest), the nominal-aware tableau always has to consider the potential degeneration of the completion tree into a forest [Baa03b].

Blocking

There is, however, one problem with complex logics such as $\mathcal{SHOF}(\mathbf{D})$. If $\exists$ is allowed in the TBox, a straightforward tableau can run into problems. Consider for example a different completion graph $G_0 = (V_0, E_0, \mathcal{L}_0)$ with $V_0 = \{x\}$, $E_0 = \emptyset$,

$\mathcal{L}_0 = \{(x : \exists r \,.\, \top)\}$ and a TBox $= \{\exists r \,.\, \top\}$. Processing x, the tableau reasoner will determine that x needs an r successor, but has none. The tableau will now simply generate a new, anonymous (definition 4.12) r-successor, which we will call y. The resulting completion graph is

$$\begin{array}{lllll} G_0 = & (V_0, & E_0 & \mathcal{L}_0 &) \\ & (\{x, y\}, & \{r(x, y)\}, & \{(x : \exists r \,.\, \top), (y : \exists r \,.\, \top)\}) & \end{array}$$

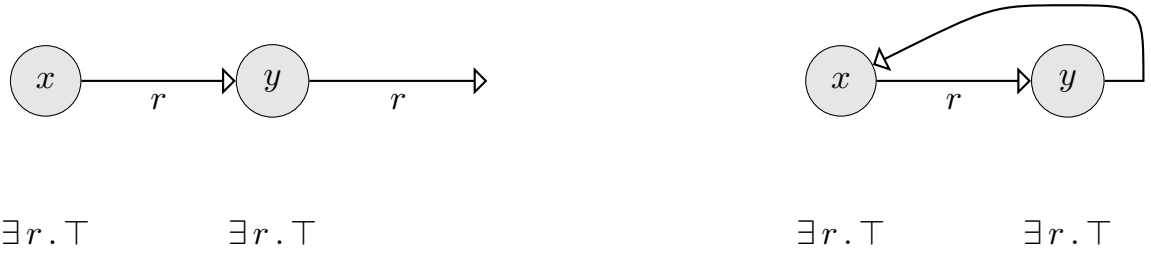
As we can see, the TBox axiom gets added to $\mathcal{L}_0[y]$ automatically: $(y : \exists r \,.\, \top)$. Now, however, while x has the r-successor demanded by the existential qualification, another successor is now demanded for y. If the tableau now also generates this additional successor z, the problem re-appears for z and the tableau would need to generate new successors ad infinitum.

Figure 4.5a illustrates the problem. After y, we would need to generate a z and after z, an infinite number of nodes would have to follow which is clearly not feasible. There are, however, other solutions to the problem. Figure 4.5b shows a saturated completion graph that is a model for the TBox-expression $(\exists r \,.\, \top)$. After generating y, it is possible to simply loop back to x. Since x in the example has at least the same associated concepts as y (i.e. $\mathcal{L}[x] \supseteq \mathcal{L}[y]$), we gain a saturated completion graph that does not clash. Since we are dealing with satisfiability checking, no further processing is necessary. Satisfiability is achieved when we find at least one saturated completion graph for the input concepts.

It is necessary to note, however, using the smallest cycle is only one possible solution (least fixed point) and that other completion graphs that satisfy the input conditions exist. As noted in [Baa03a], a cycle of any length (including infinite length) represents a model for the input concept. In general, the interpretation can be a *least fixed point* using the smallest possible cycle, it can be *greatest fixed point* and therefore infinite or it can use an arbitrary length cycle in between (*semantic interpretation* [Neb91]).

Tableau reasoners usually deal with infinite expansion by a mechanism called *blocking*. If a node is *blocked* further application of tableau rules (including the $\exists$-rule) is prevented. The condition for when a node is blocked, however, depends

Figure 4.5. Blocking in Expressive DLs



(a) Repeated Expansion due to ∃ in TBox

(b) Example of implicit least-fixed-point cycle due to blocking in expressive DLs

on the logical language used and also on a particular blocking algorithm. Usually, blocking is established as a blocking relation between another node (the blocker) and the blocked node. For description logics without inverse roles *subset blocking* [HST00a] is sufficient. If inverse roles are supported, *equality blocking* [HST00a] must be employed.

Extensions to the logic $\mathcal{SHIF}$ need *double blocking* because of the interaction between inverse and transitive roles. Use of nominals requires *dynamic* blocking, where a blocked node can be *unblocked* as reasoning progresses. Blocking conditions also sometimes need to be different between pure TBox reasoners and ABox reasoners.

Usually, blocking requires a total order on the nodes and a node can only be blocked by a blocker node that is smaller that the blocked individual. Nominal nodes cannot be blocked and blocking also requires that there are no nominal nodes between the blocked node and its blocker ancestor. These facts are –unfortunately– rarely stated explicitly in the literature.

Besides this, [HST00a] contains an in-depth treatment on the subject of blocking and DL reasoning in general.

4.2.2. Reasoner Optimizations

Because worst case performance of any reasoning algorithm with expressive description logics is in EXPTIME, increasing the performance of tableau reasoners for the "typical" ontology is subject to ongoing research. Many interesting and effective techniques have been developed to speed up reasoning performance. An overview of optimization algorithms for DL tableaux can be found in [BHN+94, HST99, Hor97, THPS07].

Causes for slow performance

A single completion graph usually grows only polynomially with the size of the input concept, but $\sqcup$-branching (or-branching) can result in a combinatorial explosion of the search space (= number of completion graphs). The main reason for this are a form of typical TBox concepts named *general concept inclusion*s. A general concept inclusion (GCI) is a TBox axiom of the form $C \sqsubseteq D$, where both C and D are non-atomic concepts. The logical language usually does not directly support such inclusion axioms (see e.g. figure 4.3), but transforms a GCI into a disjunction $(C \sqsubseteq D) \equiv (\neg C \sqcup D)$. Because the TBox-GCI will be inserted into $\mathcal{L}[x]$ for every node x, even a small number of such general concept inclusions will cause a large number of invocations of the $\sqcup$-rule and thus a large number of completion graph branches that need to be traversed individually. GCI-branching is the most significant performance problem that is faced by all tableau reasoners. A number of techniques have been developed to reduce the number of GCIs.

Domain and Range

Many of the GCIs in a typical ontology stem from *domain* assertions for properties. A domain assertion is of the form $\exists r \,.\, \top \sqsubseteq C$, which imposes a restriction on the individuals that can appear at the left side (starting point) of role connections. A typical domain model will contain many of such global axiom in its TBox. A performance improvement for role domains (and ranges) is proposed in [TH04]. Instead of axiomatizing domain and range restrictions using existing DL

Table 4.2. Tableau Rules for Absorbed Range and Domains

$domain$**-rule**	**if**	1. x is not blocked 2. $\exists\, y \in V, r \in \mathcal{R}\,.\, r(x,y) \in E$
	then	for all roles $r \in \{r \mid \exists\, y \in E, q \in r{\downarrow}\,.\, q(x,y) \in E\}$ $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(x : \mathsf{Domain}(r))\}$
$range$**-rule**	**if**	1. x is not blocked 2. $\exists\, w \in V, r \in \mathcal{R}\,.\, r(w,y) \in E$
	then	for all roles $r \in \{r \mid \exists\, w \in V, q \in q{\downarrow}\,.\, q(w,x) \in E\}$ $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(x : \mathsf{Range}(r))\}$

axioms, the Web Ontology Language provides facilities to explicitly assert the domain ($\exists\, r\,.\, \top \sqsubseteq D$) and ranges ($\forall\, r\,.\, R$) of roles. All domain and range assertions are collected and two functions are provided such that $\mathsf{Domain}(r) : \mathcal{R} \mapsto \Sigma_{L(\mathcal{I},\mathcal{C},\mathcal{R})}$ is a function returning the domain concepts for a role r and that $\mathsf{Range}(r) : \mathcal{R} \mapsto \Sigma_{L(\mathcal{I},\mathcal{C},\mathcal{R})}$ is a function returning the range concepts (or datatype ranges) for a role r. The tableau is then augmented with two additional rules (table 4.2) that assign the domain concepts to any node that has an outgoing role successor and the range concepts to any node that has an incoming role with an associated range.

Lazy Unfolding and Absorption

Another performance improvement is based on the realization that a well-designed ontology will contain only few GCIs $C \sqsubseteq D$ where D is a complex concept. Transforming this concept would lead to two branched models containing $\neg C$ and D, respectively (branched from $\neg C \sqcup D$). Branching is now always necessary, however. For certain, common types of GCIs, branching can be avoided altogether by making use of *lazy unfolding*.

We say that a concept definition $A \sqsubseteq D$ *directly uses* a concept D if D appears at the right side of the inclusion axiom. A definition *uses* a concept C if C is directly used or if any of the used concepts use C [BHN+94]. This defines the "uses" relation as the transitive closure as the *direct uses* relation (definition 4.13).

Definition 4.13 Concept Use

Given a DL TBox, the definition of A **directly uses** a concept D iff there is an axiom $(A \sqsubseteq \phi) \in$ TBox such that D is referenced in ϕ.
The concept definition of A **uses** another concept D if A directly or or transitively uses D.

The uses relation can be cyclic, that is A can use A in its own definition. For many concept definitions, however, this is not the case. Such "simple" concept definitions are called *definitorial*.

Definition 4.14 Definitorial Concept Inclusion

An DL axiom $A \sqsubseteq B$ is called **definitorial** iff $A \in \mathcal{C}$ is a primitive, named concept and the definition of A does not use A.

Definitorial general concept inclusions need not be added to the axiom set of every node inside the completion graphs. Instead, it is possible to add them *lazily* only whenever there is suitable evidence that the addition is required. This technique is known as *lazy unfolding*.

The operating principle of lazy unfolding is simple. TBox-terms of the form $A \sqsubseteq D$ are initially ignored, if A is a primitive concept. However, if A is added to a node, the list of concept inclusions is searched and D is added if $A \sqsubseteq D$ is found within the TBox. Instead of transforming the implication into negation normal

form, it is kept as is and the consequent (D) of the implication is added to a term set if its precondition (A) is already met. Lazy unfolding is only sound when A is definitorial.

Procedure 4.1: unfold(G, TBox)

```
1 Input: G = (V, E, L)          a completion graph
         TBox ⊆ L(J, C, R)      a set of TBox terms
  begin
2   foreach x ∈ E do
3     if (A ⊑ B) ∈ TBox A ∈ L[x] and A is definitorial then
4       L[x] ← L[x] ∪ {B};
      end
    end
  end
```

With lazy unfolding, branching for a majority of concept definitions (which are commonly acyclic) can be avoided altogether. Lazy unfolding is a very powerful technique and is universally used. It is usually combined with suitable *absorption* rules. In this context, absorption refers to the fact that some axioms can be removed (i.e. *absorbed*) from the TBox if they are candidates for lazy unfolding.

For example, an absorption algorithm can pick up all definitorial axioms from the TBox [HT00]. It can also perform simplifications on TBox axioms to enable their absorption. The axiom $\neg C_0 \sqcup C_1 \sqcup C_2 ...$ can be rewritten to the definitorial axiom. $C_0 \sqsubseteq (C_1 \sqcup C_2 ...)$ if C_0 is primitive. Similar rewrites are possible for $(C_0 \sqcup C_1) \sqsubseteq C_2$. Tsarkov et al. give a variety of absorption techniques used in the FacT++ Reasoner [TH04].

Boolean Constraint Propagation

Other reasoning optimisation techniques try to influence the $\sqcup$-branching operations directly. One possibility is to use heuristic sorting to order the branching operations, i.e. to perform branching on some $\sqcup$-terms before others. A very simple but effective technique is *boolean constraint propagation* (BCP). Consider a node x with

$$\begin{aligned}\mathcal{L}[x] =&\{A_0 \sqcup A_1 \sqcup C_k, B_0 \sqcup B_1 \sqcup C_{k+1},\\ &C_1 \sqcup C_2 \ldots \sqcup C_n, \neg C_1, \ldots, \neg C_{k-1}, \neg C_{k+1}, \ldots, \neg C_n\}\end{aligned}$$

A naive implementation would first branch on $A_0 \sqcup A_1 \sqcup C_k$, then on $B_0 \sqcup B_1 \sqcup C_k$ and finally on $C_1 \sqcup C_2 \ldots \sqcup C_n$. For the last branch, all but the single instantiation C_k is inconsistent. This instantiation, however, also invalidates the preconditions for all the other branches, so that we could have avoided branching altogether. BCP tries to find the union concept with the least number of yet unresolved concepts and branch this one first. The implementation of BCP for description logics has been described e.g. in [Hor97].

Semantic Branching

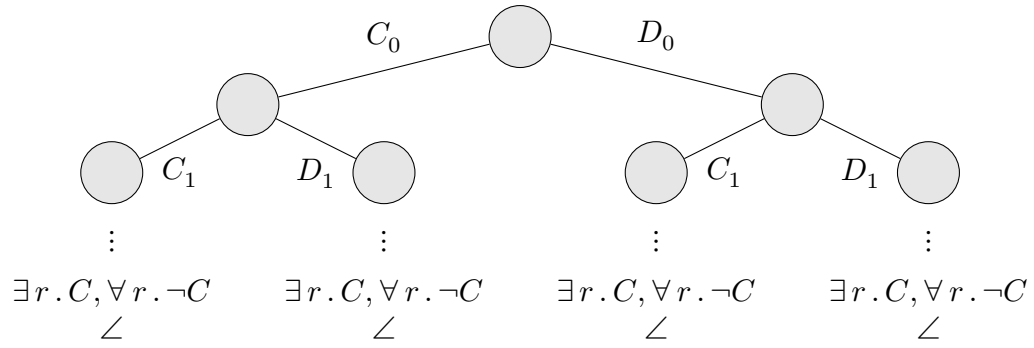
When branching cannot be avoided, techniques can be applied that try to keep the number of branch points low. A completion graph G is discarded from the branch queue $\mathbb{G}$, when it is determined, that G is inconsistent (contains a clash). It is therefore highly desirable to find inconsistencies early. One technique involves a modification of the $\sqcup$-rule. Instead of two successor graphs, three graphs are created as successors to a union branch. The concept union $\{A \sqcup B\}$ is not replaced by instances of $\{A\}$, $\{B\}$, but rather by three independent instances of $\{\neg A, B\}$, $\{A, \neg B\}$, $\{A, B\}$. This modification is known as *semantic branching* [Hor97].

Semantic branching works best when combined with BCP. Its effectiveness, however, is problem dependent and reasoning performance may actually degrade for certain ontologies.

Dependency Directed Backtracking

Another technique involves pruning branches from the branch tree after a clash has been found. Consider for example a node x with

Figure 4.6. Tableau Thrashing without Dependency Directed Backtracking



A naive tableau implementation has to saturate 2^n tableau branches before it detects that the input concept is not satisfiable

$$\mathcal{L}[x] = \{C_0 \sqcup D_0, \ldots, C_n \sqcup D_n, \exists r\,.\,C \sqcup D, \forall r\,.\,\neg C\}$$

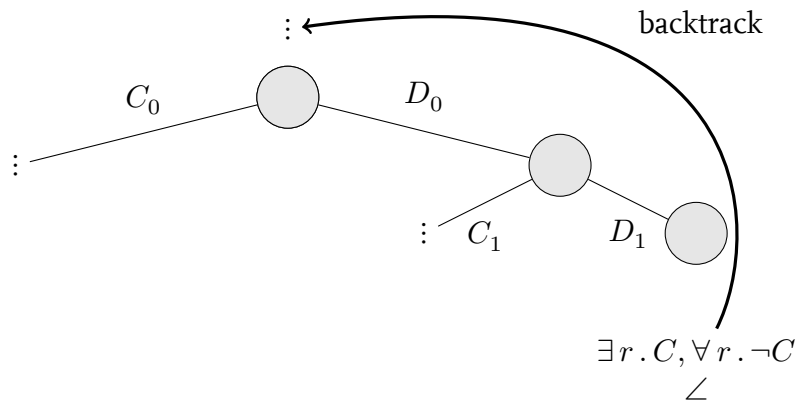
A naive algorithm might branch on the various disjoints (causing 2^n branches) only to find out that the initial concept cannot be satisfied regardless of the branch chosen for all of the unions (figure 4.6). Dependency directed backtracking records the clashing concepts and jumps back to a branching point before the clashing concepts where introduced, effectively pruning the other branches between clash detection and the introduction of the culprit term. In the example, the algorithm would report an inconsistency after the first clash is found, because the clashing terms already exist in the input concept.

Dependency directed backtracking is even more effective for well-designed ontologies with many GCIs, because well-designed ontologies make use of concept disjointness [RDH+04] whenever appropriate.

4.3 LillyTab

As part this thesis, a special purpose tableau reasoner "LillyTab" has been implemented. LillyTab's tableau is based on the well known tableau for $\mathcal{SHOIQ}(\mathbf{D})$ [HS05b], but is limited to the subset $\mathcal{SHOF}(\mathbf{D})$: Support for number restrictions is missing in LillyTab, since these are not needed to handle the existing built heritage document ontologies. The results from this thesis, however, do generalize

Figure 4.7. Dependency directed backtracking



Dependency directed backtracking records the clashing concepts and jumps back to a branching point before the clashing concepts where introduced, effectively pruning the other branches between clash detection and the introduction of the culprit terms

to description logics with number restrictions. Generalization to logics with inverse roles are deemed possible, but no attempt is made in this thesis. Formal semantics for $\mathcal{SHOF}(\mathbf{D})$ are given in definition 4.15.

LillyTab's tableau makes no mention of attributes ($\mathcal{A}$) and data values ($\mathcal{T}$, see definition 3.1). It is common for tableau introductions to ignore datatype reasoning as not to add unnecessary complexity to the tableau description. This practice will be repeated for the description of the LillyTab tableau.

Data values are represented as individuals and datatypes are implemented as *concrete domains* [BS03]. Reasoning with concrete domains can be reduced to node-local consistency checking without modification of the DL tableau ([Lut03, LM05]).

For every data type or data range, a unique concept D is introduced. D contains exactly those individuals that are part of the data range. For example $D_{\mathbb{N}} \equiv_{\text{def}} \{\underline{1}, \underline{2}, ...\}$ is the set of positive integer values. The reasoner does not need to represent data type sets implicitly. Instead, only two methods need to be provided per data type:

- A method to test if a particular individual $\underline{a} \in D$.
- A method to test if two individuals $\underline{a} \in D$ and $\underline{b} \in D$ do are equivalent with respect to D, i.e. if $\{\underline{a}, \underline{b}\}$ does not create a clash.

Definition 4.15 Formal Semantics for $\mathcal{SHOF}(\mathbf{D})$

Given a set of named concepts $\mathcal{C}$, a set of roles $\mathcal{R}$, and a set of individuals $\mathcal{I}$, the interpretation of a formula $\phi \in \Sigma_{L_{\mathcal{SHOF}(\mathbf{D})}}$ is defined via an **interpretation function** $\iota \equiv_{\text{def}} \iota_{\mathcal{I}} \cup \iota_{\mathcal{R}} \cup \iota_{\mathcal{C}}$ with

- $\iota(\{\underline{a}\}) = \iota_{\mathcal{I}}(\{\underline{a}\}) = \underline{a}$ for all $\underline{a} \in \mathcal{I}$
- $\iota(C) = \iota_{\mathcal{C}}(C) \subseteq \mathcal{I}$
- $\iota(r) = \iota_{\mathcal{R}}(r) \subseteq \mathcal{I} \times \mathcal{I}$

such that for each subformula it holds recursively

- $\iota(C \sqcap D) = \iota(C) \cap \iota(D)$
- $\iota(C \sqcup D) = \iota(C) \cup \iota(D)$
- $\iota(\neg C) = \mathcal{I} - \iota(C)$
- $\iota(\forall\, r\,.\, C) = \{x \in \mathcal{I} | \forall\, y \in \mathcal{I}\,.\,(x, y) \in \iota(r) \Rightarrow y \in \iota(C)\}$
- $\iota(\exists\, r\,.\, C) = \{x \in \mathcal{I} | \exists\, y \in \mathcal{I}\,.\,(x, y) \in \iota(r) \wedge y \in \iota(C)\}$

together with the RBox constraints transitive(r), functional(r), and $r \sqsubseteq q$.

4.3.1. Rationale

Implementing a DL reasoner is a complex and cumbersome process. Implementing a DL reasoner as part of a research effort should therefore be considered very carefully. At the time when LillyTab's implementation was started, several off-the-shelf reasoners like Racer, Pellet, and Fact++ were already available. However, none of those and also none of the current reasoning systems are built to be extensible. Most reasoners use a highly optimized internal representation of completion graphs. This optimized representation makes it is hard to easily follow the path of an axiom as it is transformed by the reasoner.

Also, reasoners only rarely provide the ability to selectively enable/disable certain optimizations.

LillyTab provides these features. And while it may not be as fast as the more optimized reasoning systems and it also does not cover more complex features of modern DLs, it has already been proven as an adaptable reasoning system, that is easy to get familiar with and extend.

4.3.2. Tableau Rules

With the basic notions of the tableau established, we can proceed to the description of the actual tableau implementation. LillyTab's tableau is an implementation of the $\mathcal{SHOIQ}(\mathbf{D})$ tableau from [HS05b], but with those features removed that are not in $\mathcal{SHOF}(\mathbf{D})$.

A single *macro operation* merge is defined (function 4.2). merge combines two individuals from a completion graph into a single individual, preserving all connections and associated concepts. A similar merge operation is described in [HS05b]. The version presented here is semantically equivalent, but operates on slightly different data structures.

Function 4.2: merge(G, t, s)

Input: $G = (V, E, \mathcal{L})$ a DL completion graph
$s \in E$ the source node of the merge
$t \in E$ the target node of the merge
Output: $G_m = (V_m, E_m, \mathcal{L}_m)$
begin
1 **if** *t is anonymous and s is not anonymous* **then**
2 $G_m \leftarrow \mathsf{merge}(G, t, s)$;
else
3 $V_m \leftarrow V - \{s\}$;
4 $E_m \leftarrow (E - \{r(x, y) | x = s \vee y = s\})$
$\cup \{r(x, t) | r(x, s) \in E\}$
$\cup \{r(t, y) | r(s, y) \in E\}$;
5 $\mathcal{L}_m \leftarrow (\mathcal{L} - \{(s : C) \,|\, (s : C) \in \mathcal{L}\})$
$\cup \{(t : C) \,| C \in \mathcal{L}[s]\}$;
6 $G_m \leftarrow (V_m, E_m, \mathcal{L}_m)$;
end
7 **return** G_m;
end

As a short form, a sequence of merges will be written using a syntax similar to variable substitution (appendix A.1):

$$\begin{aligned} G\left[t \leftarrow s\right] &= \mathsf{merge}(G, t, s) \\ G\left[t \leftarrow s, x \leftarrow y\right] &= G\left[t \leftarrow s\right]\left[x \leftarrow y\right] \\ &= \mathsf{merge}(\mathsf{merge}(G, t, s), y, x) \end{aligned}$$

The tableau rules for LillyTab are shown in tables 4.3 and 4.4. The $\sqcap$-, $\sqcup$-, and $\forall$-rules are called *non-generating* rules, as they do not create new nodes or modify the graph beside adding concepts to individual nodes. The $\exists$-rule is called a *generating* rule. The o- and $\mathcal{F}$-rule are called *merge rules*, since they invoke the merge operation. Most of the completion rules typically pick up a complex concept from the completion graph and transform it into one or more simpler concepts or add a node and/or links to the completion graph. Consequently, the completion graph usually gets larger during tableau completion, but there are two rules (o-rule, $\mathcal{F}$-rule) that *shrink* the completion graph by merging two nodes. Special attention needs to be applied to these rules as they could potentially cause non-termination.

LillyTab assumes that all concepts in $\mathcal{L}$ are automatically transformed to NNF.

LillyTab applies rules in a certain order. This application order is mandated by the original tableau [HS05b]. Rule application is performed for every non-saturated and non-clashing graph, as long as $\mathbb{G}$ still contains non-saturated graphs (clashing graphs are automatically removed).

LillyTab's tableau does not implement some of the rules from the initial tableau, because the full expressiveness of $\mathcal{SHOQ}$ is not required. It is important to make sure that the removed rules do not affect the properties (correctness, soundness, termination) of the tableau with regard to the reduced language subset $\mathcal{SHOF}(\mathbf{D})$.

Theorem 4.1 Soundness, Completeness, and Termination of the LillyTab Tableau for $\mathcal{SHOF}(\mathbf{D})$

The rules in tables 4.3 and 4.4 are a sound, complete, and terminating tableau decision procedure for $\mathcal{SHOF}(\mathbf{D})$

Table 4.3. Tableau Rules for the LillyTab Reasoner, part 1

Given a set $\mathbb{G}$ of DL completion graphs $\mathbb{G} = \{G_1, \ldots, G_m\}$, for each graph $G_k \in \mathbb{G}$ and each node $x \in G_k$, repeat the following rules until no rule is applicable any more:

⊓-rule **if** 1. x is not blocked
2. $\exists\, C, D \,.\, (x : C \sqcap D) \in \mathcal{L}_k$
3. not $\{(x : C), (x : D)\} \subseteq \mathcal{L}_k$

then update $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(x : C), (x : D)\}$

⊔-rule **if** 1. x is not blocked
2. $\exists\, C, D \,.\, (x : C \sqcup D) \in \mathcal{L}_k$
3. $\{(x : C), (x : D)\} \cap \mathcal{L}_k = \emptyset$

then Create two fresh successor graphs

$G_{k+1} \leftarrow G_k$, $\quad G_{k+2} \leftarrow G_k$,
$\mathcal{L}_{k+1} \leftarrow \mathcal{L}_k \cup \{(x : C)\}$, $\quad \mathcal{L}_{k+2} \leftarrow \mathcal{L}_k \cup \{(x : D)\}$,

$$\mathbb{G} \leftarrow (\mathbb{G} - G_k) \cup \{G_{k+1}, G_{k+2}\}.$$

∀-rule **if** 1. x is not blocked
2. $\exists\, r, C \,.\, (x : \forall\, r \,.\, C) \in \mathcal{L}_k$

then for all $y \in V_k, q \in r{\downarrow}$ with $q(x, y) \in E_k$
set $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(y : C)\}$

∀⁺-rule **if** 1. x is not blocked
2. $\mathsf{transitive}(r)$
3. $\exists\, r, C \,.\, (x : \forall\, r \,.\, C) \in \mathcal{L}_k$

then for all $y \in V_k, q \in r{\downarrow}$ with $q(x, y) \in E_k$
set $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(y : \forall\, r \,.\, C)\}$.

∃-rule **if** 1. x is not blocked
2. $(x : \exists\, r \,.\, C) \in \mathcal{L}_k$
3. $\nexists y \in V_k, q \in r{\downarrow} \,.\, q(x, y) \in E_k \wedge (y : C) \in \mathcal{L}_k$

then if $\mathsf{functional}(r)$ and $\exists\, y \,.\, r(x, y) \in E_k$, then set $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(y : C)\}$.
else obtain a fresh, anonymous individual $y \in N$ and set $V_k \leftarrow V_k \cup \{y\}$, $E_k \leftarrow E_k \cup \{r(x, y)\}$, and $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(y : C)\}$

Table 4.4. Tableau Rules for the LillyTab Reasoner, part 2

Given a set $\mathbb{G}$ of DL completion graphs $\mathbb{G} = \{G_1, \ldots, G_m\}$, for each graph $G_k \in \mathbb{G}$ and each node $x \in G_k$, repeat the following rules until no rule is applicable any more:

$\mathcal{F}$**-rule**	**if**	1.	x is not blocked
		2.	functional(r)
		3.	$\exists\, y, z\,.\,\{r(x,y), r(x,z)\} \subseteq E_k \wedge y \neq z$
	then		$G_k \leftarrow$ merge (G_k, z, y)
o**-rule**	**if**	1.	x is not blocked
		2.	$\exists\, \underline{a} \in \mathcal{I}\,.\,\{\underline{a}\} \in \mathcal{L}_k[x]$
	then		$G_k \leftarrow$ merge $(G_k, \underline{a}, x)$

Definition 4.16 Priority of Rule Application

1. o- and $\mathcal{F}$-merge are executed immediately when their preconditions are satisfied.
2. All possible $\sqcap$, $\sqcup$, $\forall$ rules are performed until none of these rules remains applicable.
3. If an $\exists$-rule is applicable, only a single application of $\exists$ is performed.

This process is repeated until no more rules have their preconditions met or until a clash is detected.

Theorem 4.1 is proven in appendix A.2.

LillyTab checks the completion graph for inconsistencies after each cycle of rule application. Consistency checking for concrete domains (datatypes) in LillyTab is provided by the introduction of unique concepts $D_0, D_1, ...$ for each datatype range. If $(x : D_i) \in \mathcal{L}$, then the node x is assumed to have datatype D_i. Each data range concept has a fixed (usually infinite) interpretation domain $\iota_{\mathcal{D}}(D_i)$. A named node is verified to be within the interpretation domain of each of its annotated data ranges. Beyond this, the consistency conditions are the same as for $\mathcal{SHOQ}$ with those removed that can never be violated by a $\mathcal{SHOF}(D)$ completion graph:

Definition 4.17 Consistency Checks for $\mathcal{SHOF}(D)$

Given a (saturated) completion graph $G = (V, E, \mathcal{L})$ over the language $L_{(\mathcal{SHOF}(\mathbf{D}),\mathcal{I},\mathcal{C},\mathcal{R})}$, a node $x \in E$ is said to contain a **clash**, iff

1. for some concept $C \in \Sigma_L$ $\{C, \neg C\} \subseteq \mathcal{L}[x]$, or
2. functional$(r) \wedge \exists y, z\{r(x, y), r(x, z)\} \subseteq E$, or
3. x is annotated with data ranges $\{D_i, D_j\} \subseteq \mathcal{L}[x]$ and $\iota_{\mathcal{D}}(D_i) \cap \iota_{\mathcal{D}}(D_j) = \emptyset$, i.e. the intersection of the two data ranges is empty, or
4. $\{\underline{a}\} \in \mathcal{L}[x] \wedge D \in \mathcal{L}[x] \wedge \underline{a} \notin \iota_{\mathcal{D}}(D)$ for some individual $\underline{a}$ and some data range D, i.e. one of the individuals of a node is not part of one of the data ranges declared for that node, or
5. if $\{\{\underline{a}\}, \{\underline{b}\}\} \subseteq \mathcal{L}[x] \wedge D \in \mathcal{L}[x]$, $\underline{a}$ and $\underline{b}$ are incompatible with regard to D, i.e. they represent conflicting values with regard to D.

If at least one node $x \in E$ contains a **clash**, the completion graph is said to be **inconsistent**.

4.3.3. Tableau Data Structures

The basic data structures in LillyTab are a set $\mathbb{G}$ of completion graphs and the TBox. The TBox is simply a set of global axioms that are valid for every node inside a completion graph. When a fresh node x is created inside a completion graph, it is therefore assumed that $\mathcal{L}[x] \supseteq$ TBox. This is different from the initial

$\mathcal{SHOQ}$ tableau. In the $\mathcal{SHOQ}$-tableau the TBox is represented by restrictions on a universal rule $\mathcal{U}$. $\mathcal{U}$ is assumed to connect every individual node with every other individual node. Consequently, asserting $\forall\, \mathcal{U}\, .\, C$ for some individual x in the completion graph automatically propagates C to every graph node (including x, since $\mathcal{U}$ is reflexive). LillyTab uses a direct TBox for efficiency reasons. There is no semantic difference to the $\mathcal{U}$-approach found –for example– in the $\mathcal{SHOQ}$ and RACE [HM01] tableaux.

The basic operation of LillyTab is similar to that of other tableau reasoners. The tableau starts with an initial set of completion graphs $\mathbb{G}$ and a set of tableau rules. The tableau rules transform the initial graphs such that satisfiability is maintained and eventually a point is reached, where

1. either no more rules can be applied to any graph, including the case when G is empty, or

2. a consistent graph is found where no more rules can be applied to the graph.

The state where no more rule application is possible, is called *saturated*. Rule application to a graph also stops, when a clash is detected in a graph. Graphs that contain a clash are automatically removed from $\mathbb{G}$.

LillyTab is different from other reasoners, as it can be switched into a mode where finding a saturated graph does not terminate reasoning. In this mode, rather than returning the first saturated graph, LillyTab continues until all possibilities have been explored. The refinement process detailed in the later parts of this thesis makes heavy use of this unique feature.

Definition 4.18 Saturated Completion Graph

A DL completion graph G is called **saturated** wrt. to a particular tableau, if none of the tableau's transformation rules can be applied to the graph.

4.3.4. Reasoner Implementation

Blocking Implementation

Since LillyTab only supports the logic $\mathcal{SHOF}(\mathbf{D})$, it is sufficient to apply only subset blocking ([HST99], definition 4.19). Blocking in LillyTab is dynamic. Because LillyTab contains support for nominals (but not for inverse roles), the axiom set of a blocker may be modified again during tableau operation. Since this may invalidate the block (i.e. a new concept might be introduced), any modification of the axiom set of the *blocker* node requires a re-evaluation of the block. LillyTab explicitly keeps track of the blocker-blockee relationship and re-evaluates a blocking condition when either the blocker or the blocked node are modified.

The tableau rules (see section 4.3.2, below) never modify a blocked node directly. Consequently, the need for re-checking a blocked node rarely occurs, making blocking state caching very effective.

Definition 4.19 Subset Blocking

Let $G = (V, E, \mathcal{L})$ be a completion graph.

- y is an *predecessor* of x iff $\exists\, r\,.\, r(y, x) \in E$.
- z is an *ancestor* of x iff it is in the transitive predecessor-closure of x.
- w is an anonymous node iff $\nexists \underline{a}\,.\,\{\underline{a}\} \in \mathcal{L}[w]$

Some node x is *subset blocked*, if there is some $b \in V$, such that

1. b is an anonymous node.
2. b is an ancestor of x
3. $\mathcal{L}[x] \subseteq \mathcal{L}[b]$

b is then called a **blocker** of x.

Implemented Optimizations

LillyTab implements a number of optimizations to speed up the reasoning process. In particular:

Domain and Range Absorption LillyTab implements role domain and range absorption in line with the modified tableau from [TH04] (table 4.2).

Concept Absorption LillyTab implements concept absorption and applies several simplification rewrites on input TBoxes to improve reasoner performance.

Semantic Branching LillyTab can optionally perform semantic branching but does not by default.

Dependency Directed Backtracking LillyTab implements dependency directed backtracking.

Lazy Saving LillyTab implements lazy saving as copy-on-write on the concept and role list data structures.

SECTION 5.0

Completion Graph Based Mapping

In this chapter, a correspondence between conjunctive queries and DL completion graphs is established. This creates a framework for using completion graphs as a representation method for directional mappings between ontologies.

5.1 Completion Graphs and Queries

While simple mappings can be expressed as a quadruple (e_s, e_t, r, n) (see section 3.2), complex ontology matching requires the ability to both express queries on the source ontology as well as to formulate *generating expressions* for creating individuals and literals in the target ontology.

One interesting property of DL completion graphs is, that it is possible to view them both as a pseudo-model as well as a generation and classification scheme for ABox individuals: anonymous nodes in the completion graph represent any individual that fits the constraints in the node's associated axiom set. Named nodes (those labelled with a nominal $\{\underline{a}\}$) represent exactly the named individual. The same is true on the generating side, with the difference, that anonymous nodes on the target side represent either any *existing* individual meeting the desired constraints or a fresh node (x) that needs to be generated.

Consider the completion graph

$$\begin{array}{rllll} G_0 = & (V_0, & E_0, & \mathcal{L}_0 &) \\ = & (\{x, y\} & \{r(x, y)\}, & \{(x : A), (y : \{\underline{b}\}), (y : B)\}) \end{array} \tag{5.1}$$

Turning to the interpretation of this graph (using definition 4.15), we find that

1. $(y : \{\underline{b}\}) \in \mathcal{L}_0$ pins y to be equivalent to the individual $\underline{b}$.
2. $(x : A) \in \mathcal{L}_0$ demands x to be in the interpretation $\iota(A)$ of A and $(y : B) \in \mathcal{L}_0$ requires that y is in the interpretation of B. Together with (1), this means, that $B(\underline{b})$ and $y = \{\underline{b}\}$. x can be selected arbitrarily, while y is pinned to $\{\underline{b}\}$.
3. $(r(x, y))$ requires an r-connection between x and y.

If we replace the names of anonymous nodes with variable names ($?x$) and use the nominal reference in place of y, we can directly formulate the interpretation of G_0 in first order logic

$$A(?x) \wedge r(?x, \underline{b}) \wedge B(\underline{b}) \tag{5.2}$$

where $?x$ is a free variable, so this formula can also be seen as a query that returns $?x$. If this query is performed on a DL knowledge base, it returns exactly those individuals (in $?x$) that can be placed into the position of x in the completion graph and satisfy the constraints established by the graph.

This simple example shows, that completion graphs can indeed be viewed as templates to express queries on DL knowledge bases. This makes it possible to consider complex ontology mapping (involving queries on knowledge bases) in terms of completion graphs and vice versa.

5.1.1. Mapping Conjunctive Queries to Completion Graphs

If we allowed for first order formulæ in mapping rules, this would raise the problem of decidability. What is needed is –once more– a decidable subset of first order logic that is nonetheless sufficiently expressive.

Queries like those presented in equation (5.2) follow a structure that is very well known from relational databases. Queries of this form are called *conjunctive queries* [Var82]. It is already known that conjunctive queries and DL concept descriptions

have a strong relationship. For example, it is possible to map conjunctive queries with a single free variable to $\mathcal{ALN}$ concept descriptions as long as the query is a tree query [GR02].

Definition 5.1 Conjunctive Query

Let X be a set of variables and $X_f \subseteq X$, $Z_i \subseteq (X_f \cup \mathcal{I})$, ... be distinct sets of variables and/or individuals and P_i be n-ary boolean predicates
A **conjunctive query** Q is an expression of the form

$$Q : \exists X_e \,.\, \bigwedge_i P_i(Z_i)$$

The variables $X_f \equiv X - X_e$ are called the *free variables* or *query result* of the query, while X_e are the bound variables of the query.
$P_0(Z_0) \wedge P_1(Z_1)$ ... is also written as $P_0(Z_0), P_1(Z_1), ...$

A first observation in this context is, that every conjunctive query has one or more associated completion graphs that are semantically equivalent to the query. This means that it is possible to convert any conjunctive query Q into a DL completion graph that covers exactly those individuals that are also covered by the query. To facilitate the conversion, we first define a mapping function σ_q that transforms variables from Q into anonymous nodes in G_q and individual references in Q into appropriately named nodes in G_q. This mapping is trivially possible as we can simple re-use variable and individual names:

The mapping is only possible if G_q has exactly as many nominal references as there are individual references in Q and the same for variables. For simplicity, we assume that the *unique name assumption* holds and that consequently G_q only contains nodes associated with at most one nominal. The unique name assumption simplifies the presentation of all subsequent algorithms. If the unique name assumption does not hold, all subsequent references to $\sigma_q^{-1}(x)$ are implicitly assumed to iterate over all nominal references of a particular node x.

Having obtained a mapping between the variables in a query and the anonymous nodes in a completion graph, it is possible to construct a completion graph G_q that has the same interpretation of the query Q. Given the description logic $L(\mathcal{I}, \mathcal{C}, \mathcal{R})$

Definition 5.2 Query Node Mapping

Given a description logic language L_{DL} and a conjunctive query

$$Q = \exists X_e \,.\, \bigwedge_i P_i\,(Z_i)$$

with $P_i \in (\mathcal{C} \cup \mathcal{R})$, and a completion graph $G_q = \left(E_q, V_q, \mathcal{L}_q\right)$
A function $\sigma_q : X \mapsto E$ is a **query node mapping**, iff

1. $\forall\, ?x \in \bigcup_i Z_i \,.\, \exists_1 x \in V_q \,.\, \sigma_q(?x) = x \wedge \left\{\underline{a} | \{\underline{a}\} \in \mathcal{L}_q[x]\right\} = \emptyset$,
2. $\forall\, \underline{a} \in \bigcup_i Z_i \,.\, \exists_1 x_{\{\underline{a}\}} \in V_q \,.\, \sigma_q(\underline{a}) = x_{\{\underline{a}\}} \wedge \{\underline{a}\} \in \mathcal{L}_q[x_{\{\underline{a}\}}]$, and
3. σ_q is bijective.

The inverse of σ_q is named σ_q^{-1}.

and a conjunctive query $Q = \exists X_e \,.\, \bigwedge_i P_i\,(X_i)$ with $P_i \in (\mathcal{C} \cup \mathcal{R})$, the corresponding completion graph $G_q = \left(E_q, V_q, \mathcal{L}_q\right)$ is constructed by the algorithm presented in function generateGraph 5.1.

This completion graph has the property that any individual that matches the conjunctive query also satisfies the completion graph.

Theorem 5.1 Correctness of Completion Graph Generation from Conjunctive Query

A completion graph G_q constructed from a query Q using function generateGraph (function 5.1) has in its interpretation the same knowledge base subsets that are matched by Q.

5.1.2. Extracting Queries from Completion Graphs

While converting a conjunctive query into a completion graph is straightforward, the inverse direction is not as simple. It is not possible to convert any completion graph into a conjunctive query with the same interpretation. Indeed, the expressiveness of DL completion graphs is strictly higher than that of conjunctive queries.

Procedure 5.1: generateGraph(Q)

Input: Q: a conjunctive query
Output: $G_q = (V_q, E_q, \mathcal{L}_q)$: a completion graph with the same interpretation as Q

begin

/* Construct a query node mapping σ_q from Q to G_q and fill G_q with the nodes from σ_q. */

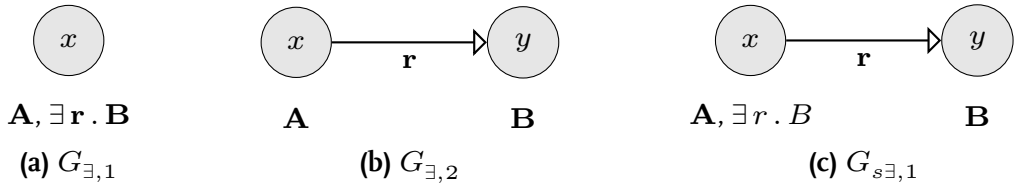
1 $\mathcal{L}_q \leftarrow \emptyset$;
2 $E_q \leftarrow \emptyset$;
3 $V_q \leftarrow \{x | x = \sigma_q(X_i)$ for all $X_i \in X\}$;
4 **foreach** $\underline{a} \in (X \cap \mathcal{I})$ *and* $\underline{a}$ *appears in* Q **do**
5 $\quad \mathcal{L}_q[\sigma_q(\underline{a})] \leftarrow \mathcal{L}_q[\sigma_q(\underline{a})] \cup \{\{\underline{a}\}\}$;
end
6 **foreach** $r(X_0, X_1) \in Q$ **do**
7 $\quad E_q \leftarrow E_q \cup \{r(\sigma_q(X_0), \sigma_q(X_1))\}$;
end
8 **return** $(V_q, E_q, \mathcal{L}_q)$;

end

The description of function 5.1 might suggest, that conversion of completion graphs to conjunctive queries is only possible if the completion graph contains only references to named concepts. This is true, if the conversion process is performed purely based on syntax alone.

When looking at the underlying semantics, however, the situation changes: two different syntactic representations may be semantically equivalent. This means, that the interpretation of a completion graph typically is not unique to that particular completion graph. Thus, if we cannot convert a completion graph into a conjunctive query directly (because it contains complex concept descriptions), we might be able to find another, semantically equivalent completion graph where the conversion is feasible.

Figure 5.1. Alternative Representations for Completion Graphs



(a) $G_{\exists,1}$ **(b)** $G_{\exists,2}$ **(c)** $G_{s\exists,1}$

Consider the completion graph $G_{\exists,1}$ in equation (5.3). Because of the $\exists$-axiom, a direct conversion is not possible, as we have no immediate way of representing the $\exists$-restriction in a conjunctive query term.

$$G_{\exists,1} = (V_{\exists}, E_{\exists}, \mathcal{L}_{\exists}) = (\{x\}, \emptyset, \{(x : A), (x : \exists\, r\, .\, B)\}) \tag{5.3}$$

If we, however, consider instead the completion graph $G_{\exists,2}$, the situation is different.

$$G_{\exists,2} = (\{x, y\}, \{r(x, y)\}, \{(x : A), (y : B)\}) \tag{5.4}$$

$G_{\exists,2}$ does not contain any complex axiom and can be mapped using a direct mapping, resulting in the conjunctive query $Q_{\exists}$.

$$Q_{\exists} = A(?x), r(?x, ?y), B(?x) \tag{5.5}$$

When we look at the interpretation of $G_{\exists,1}$ and $G_{\exists,2}$, we can immediately see that both graphs share the same interpretation. $(x : \exists\, r\, .\, B)$ in equation (5.3) is covered by the existence of the explicit successor y with its associated concept $(y : B)$ in equation (5.4). $G_{\exists,1}$ and $G_{\exists,2}$ are therefore *semantically* equivalent. Consequently, it is irrelevant if we use $G_{\exists,1}$ or $G_{\exists,2}$ as conversion source since both graphs are equivalent in their interpretation.

Figure 5.2. $\sqcap$-Propagation for Tableau Completion

(a) $G_{\exists,1}$ **(b)** $G_{\exists,2}$

Another interesting observation can be made, if we consider the saturated form (i.e. after tableau completion) of $G_{\exists,1}$:

$$G_{s\exists,1} = (\{x, y\}, \{r(x, y)\}, \{(x : A), (x : \exists\, r\,.\, B), (y : B)\}) \tag{5.6}$$

$G_{s\exists,1}$ is almost equivalent to $G_{\exists,2}$ with the difference that the initial $(x : \exists\, r\,.\, B)$-axiom is still present in $G_{s\exists,1}$ but is missing in $G_{\exists,2}$. All three graphs, however, share the same interpretation, i.e. $G_{\exists,1}$, $G_{s\exists,1}$ and $G_{\exists,2}$ (figure 5.1) are all semantically equivalent.

A similar argument can be made for $\sqcap$-axioms. Given the completion graph

$$G_{\sqcap} = (\{x\}, \emptyset, \{(x : A \sqcap B)\}) \tag{5.7}$$

its saturated equivalent is

$$G_{s\sqcap} = (\{x\}, \emptyset, \{(x : A \sqcap B), (x : A), (x : B)\}) \tag{5.8}$$

Once again, we can consider only the atomic axioms in the saturated completion graph $G_{s\sqcap}$ to obtain the proper conjunctive query $Q_{\sqcap} = A(?x), B(?x)$. A similar argument can be made for $\sqcup$-axioms. Here, however, we get two separate completion graphs (when not using semantic branching) and consequently two different query terms:

$$G_{\sqcup} = (\{x\}, \emptyset, \{(x : A \sqcup B)\}) \tag{5.9}$$

$$\begin{aligned} G_{s\sqcup,1} &= (\{x\}, \emptyset, \{(x : A \sqcup B), (x : A)\}), \quad Q_{\sqcup,1} = A(?x) \\ G_{s\sqcup,2} &= (\{x\}, \emptyset, \{(x : A \sqcup B), (x : B)\}), \quad Q_{\sqcup,2} = B(?x) \end{aligned} \tag{5.10}$$

The basic algorithm for converting completion graphs into conjunctive queries can be sketched as follows:

1. Given a completion graph, use a tableau reasoner to transform the graph into its saturated form(s). We need to expand all saturated forms.

2. Ignore all complex concept assertions and only consider atomic concept and role assertions (links) and generate appropriate query terms using a suitable query node mapping.

The simple procedure is, unfortunately, not always sufficient. It is easy to construct counter-examples that show that there still remain completion graphs that cannot be properly mapped into a conjunctive query even in their saturated versions. These graphs can be identified by the axioms that are contained in their axiom set $\mathcal{L}$:

¬-axioms Because conjunctive queries can only express positive predicates, ¬-axioms in the completion graph represent a problem. This, however, applies only to negations with a odd negation nesting.

∀-axioms Because conjunctive queries can only express existential qualification, ∀-axioms in the completion graph represent a problem. However, not all ∀-axioms are problematic.

TBox axioms During tableau reasoning, information from the TBox is integrated into the saturated graph. A non-empty TBox causes additional axioms to be added to the completion graph. Not all of these axioms need to be considered during query extraction. Which axioms are important and which are not needs to be determined.

RBox axioms Not only TBox-axioms, but also a non-empty RBox causes differences in the interpretation of a DL completion graph. Role hierarchies additionally can cause tableau completion to become non-deterministic. This non-determinism can also result in different queries being generated for the same initial completion graph, which has to be avoided.

These problems will be presented and analysed in detail in the following sections. Section 5.2 then presents a modified tableau that incorporates important changes to enable extraction of conjunctive queries from saturated DL completion graphs.

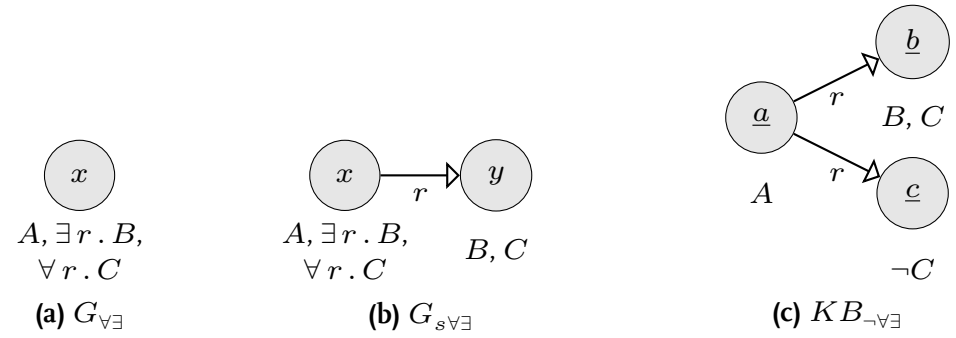
5.1.3. Handling Negation

Since conjunctive queries only support the specification of positive predicates, negation in DL completion graphs is a problem. If a completion graph contains negations in front of a primitive predicate (i.e. $\neg A$, where $A \in \mathcal{C}$), this prevents the conversion of the completion graph into a conjunctive query.

However, not all negations are problematic. If the number of nested negations is even for any nesting path in an axioms, the negation is unproblematic, as it will be cancelled out. This can easily be checked by transforming the axioms into negation normal form (definition 4.2). Negated axioms are only problematic for conversion, when their negation normal form (NNF) contains negations before primitive concepts.

In practice, unhandled negations in the final, saturated completion graph do mostly appear as consequents of already handled axioms and can thus be safely ignored.

Figure 5.3. ∀-Handling in Conjunctive Query Generation and Mismatching Knowledge Base



(a) $G_{\forall\exists}$ **(b)** $G_{s\forall\exists}$ **(c)** $KB_{\neg\forall\exists}$

$$Q_{s\forall\exists} = A(?x), r(?x, ?y), B(?y), C(?y)$$

5.1.4. Handling Universal Quantifiers

∀-axioms are problematic, because conjunctive queries do not have a concept of universal quantification. For example, when we add a ∀-restriction to the previously described $G_{\exists,1}$ (equation (5.3)), we obtain

$$\begin{aligned} G_{\forall\exists} = &\ (\ V_{\forall\exists}, \quad E_{\forall\exists}, \quad \mathcal{L}_{\forall\exists} \qquad\qquad\qquad\qquad\qquad\qquad\ ) \\ = &\ \Big(\{x\}, \quad \emptyset, \qquad \{(x : A), (x : \exists\, r\,.\, B), (x : \forall\, r\,.\, C)\}\Big) \end{aligned} \tag{5.11}$$

The saturated version of this graph is

$$\begin{aligned} G_{s\forall\exists} = &\ (\ V_{s\forall\exists}, \qquad E_{s\forall\exists}, \qquad \mathcal{L}_{s\forall\exists} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\ ) \\ = &\ \left(\{x, y\}, \quad \{r(x, y)\}, \quad \left\{ \begin{array}{l} (x : A), (x : \exists\, r\,.\, B), (x : \forall\, r\,.\, C), \\ (y : B), (y : C) \end{array} \right\}\right) \end{aligned} \tag{5.12}$$

If we apply the strategy of dropping all complex axioms from the completion graph and only converting all simple axioms, we get the query

$$Q_{s\forall\exists} = A(?x), r(?x, ?y), B(?y), C(?y) \tag{5.13}$$

While this looks fine, there is a significant difference in the interpretation of $Q_{\forall\exists}$ compared to the interpretation of $G_{\forall\exists}$. In particular, $Q_{\forall\exists}$ matches $\underline{a}$ in the knowledge base (figure 5.3)

$$KB_{\neg\forall\exists} = \{A(\underline{a}), r(\underline{a}, \underline{b}), r(\underline{a}, \underline{c}), B(\underline{b}), C(\underline{b}), \neg C(\underline{c})\} \tag{5.14}$$

$KB_{\neg\forall\exists}$ is, however, not a model for the initial completion graph. $\underline{a}$ clearly has an r-successor $\underline{c}$ with $\neg C(\underline{c})$, which contradicts $(x : \forall\, r\,.\, C)$ in the original graph. Thus, while it seems fine to drop the $\exists$-restriction, dropping the $\forall$-restriction can potentially lead to an insufficient query that is more permissive than the original completion graph.

Fortunately, the problem seems to appear rarely in practice. Most $\forall$-axioms are results from TBox expansion and thus logically dependent on some other (atomic) axiom. Also, $\forall$ axioms are only problematic for non-functional roles.

If the problem still appears, it is possible to introduce a workaround: for every axiom of the form $\forall\, r\,.\, D$ that is required to represent a completion graph (a *governing term*, the concept will be formalized in section 5.2), it is possible to introduce a fresh, previously unused concept C together with a TBox-axiom $C \sqsubseteq \forall\, r\,.\, D$ and replace all other references to $\forall\, r\,.\, D$ with C. C will now be used instead of $\forall\, r\,.\, D$, enabling the conversion while preserving semantics.

5.1.5. Handling TBox-axioms

The workaround from the previous section shows, that not every $\forall$-axiom prevents the proper conversion of a completion graph into a conjunctive query term. Consider the graph

$$\begin{aligned} G_{\mathsf{T}\forall\exists} &= (V_{\mathsf{T}\forall\exists}, \quad E_{\mathsf{T}\forall\exists}, \quad \mathcal{L}_{\mathsf{T}\forall\exists} \qquad\qquad\quad) \\ &= (\{x\}, \quad \emptyset, \qquad \{(x:A), (x:\exists\, r\,.\, B)\}) \end{aligned} \tag{5.15}$$

together with the TBox

$$\mathsf{TBox}_{\mathsf{T}\forall\exists} = \{A \sqsubseteq \forall\, r\,.\, C\} \tag{5.16}$$

The saturated version of $G_{\mathsf{T}\forall\exists}$ is equivalent to $G_{s\forall\exists}$ (equation (5.12)) and the corresponding query is also the same. The query again matches $KB_{\neg\forall\exists}$ (equation (5.14)), but now the situation is different. When combined with $\mathsf{TBox}_{\mathsf{T}\forall\exists}$, the knowledge base $KB_{\neg\forall\exists}$ is not consistent in itself, because $A(\underline{a})$ demands that also $(\forall\, r\,.\, C)(\underline{a})$ (lazily unfolded from $\mathsf{TBox}_{\mathsf{T}\forall\exists}$). This then leads to an inconsistency for $\underline{c}$. Concept axioms that are unfolded from the TBox as a result of an already existing axiom need not be considered when forming the conjunctive query.

Not every TBox axiom can be so easily ignored. We have already seen in section 4.2.1, that existential qualifiers in the TBox cause blocking and that the blocked node is a placeholder for an infinite model or a cycle of unknown size. Blocked nodes are troublesome for query generation. Consider the completion graph (see also figure 5.4a)

$$\begin{aligned} G_{\mathsf{T}\exists} &= (V_{\mathsf{T}\exists}, \quad E_{\mathsf{T}\exists}, \quad \mathcal{L}_{\mathsf{T}\exists} \qquad) \\ &= (\{x\}, \quad \emptyset, \qquad \{(x:A)\}) \end{aligned} \tag{5.17}$$

together with the TBox

$$\mathsf{TBox}_{\mathsf{T}\exists} = \{A \sqsubseteq \exists\, r\,.\, A\}\,. \tag{5.18}$$

Tableau expansion of $G_{\mathsf{T}\exists}$ will cause lazy unfolding of $\exists\, r\,.\, A$ unto x. This creates a fresh r-successor y which is then blocked from further expansion (figure 5.4b). Following the algorithm outlined above, the converted query would be

$$Q_{\mathsf{T}\exists,2} = A(?x), r(?x, ?y), A(?y) \tag{5.19}$$

However, $Q_{\mathsf{T}\exists,1}$ is neither the smallest nor the largest query that has the same interpretation as $G_{\mathsf{T}\exists}$. The smallest possible query is actually

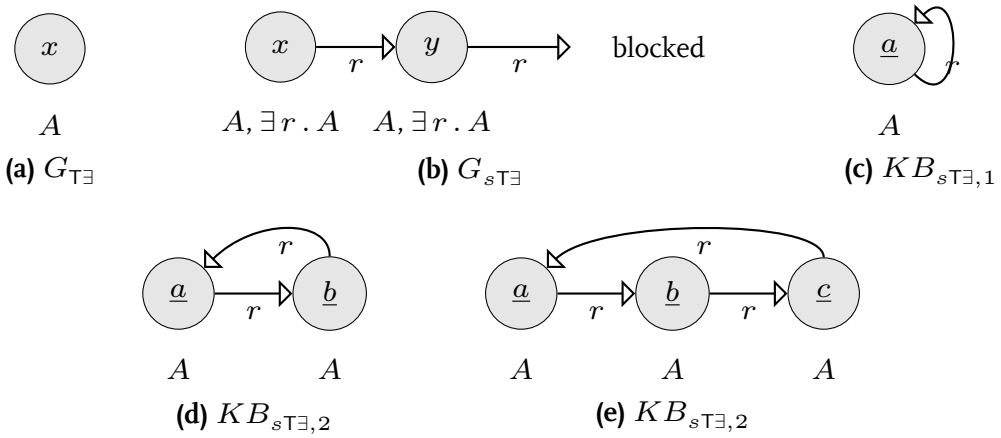
$$Q_{\mathsf{T}\exists,1} = A(?x) \tag{5.20}$$

Because designating a node with A already (via lazy unfolding of $\exists\, r\,.\, A$ from $\mathsf{TBox}_{\mathsf{T}\exists}$) starts the infinite chain of r-successors. To visualize this, look at figures 5.4c, 5.4d, and 5.4e. All the knowledge bases displayed here are models of the initial completion graph (and its corresponding) query. However, no single query can match the full extent of all possible models of a blocked completion graph. While both $Q_{\mathsf{T}\exists,1}$ and $Q_{\mathsf{T}\exists,2}$ match all knowledge bases, their coverage is different. $Q_{\mathsf{T}\exists,2}$ covers the whole of $KB_{s\mathsf{T}\exists,1}$ including the reflexive property. It also matches both individuals in $KB_{s\mathsf{T}\exists,2}$, but not the backwards reference ($r(\underline{b}, \underline{a})$). To fully cover $KB_{s\mathsf{T}\exists,2}$, we need to formulate the query

$$Q_{\mathsf{T}\exists,4} = A(?x), r(?x, ?y), A(?y), r(?y, ?z), A(?z), r(?z, ?w), A(?w) \tag{5.21}$$

It is not possible to formulate a conjunctive query that covers all possible knowledge bases that are correct interpretations of the initial model graph $G_{\mathsf{T}\exists}$. Full coverage of blocked models would require recursive queries or finite automata (see [BHP08]).

Figure 5.4. Blocked Completion Graphs and Corresponding Knowledge Bases



Two principle strategies are available for dealing with blocked nodes: 1. fail conversion when a blocked node is encountered or 2. select a suitable size for the blocking cycle and go with an appropriate query. One approach is to follow the minimal description length principle [Ris78] and generate the smallest possible query that still describes the model.

As with negations, the problem rarely exhibits itself in practice.

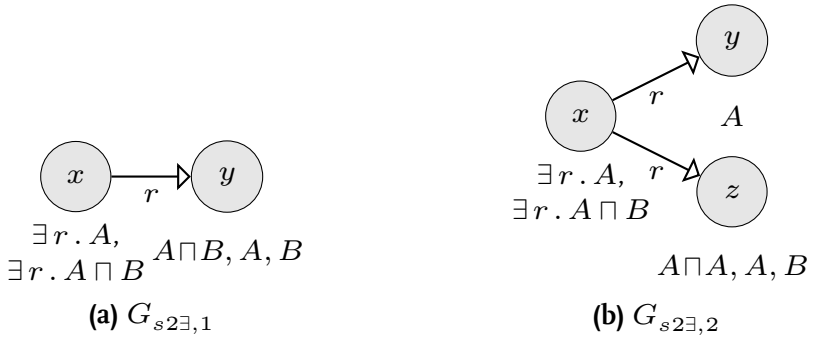
5.1.6. Handling Existential Non-Determinism

A final problem can be observed, when we consider the ∃-rule. Existing literature seems to make no note of this, but the existential expansion is actually non-deterministic.

Consider the completion graph

$$\begin{aligned} G_{2\exists} = &\ (V_{\sqsubseteq\exists}, \quad E_{\subseteq\exists}, \quad \mathcal{L}_{\subseteq\exists} \qquad\qquad\qquad\qquad\quad) \\ = &\ (\{x\}\,, \quad \emptyset, \qquad \{(x : \exists r\,.\,A)\,, (x : \exists r\,.\,A \sqcap B)\}) \end{aligned} \tag{5.22}$$

Figure 5.5. Non-determinism during Tableau Expansion with Concept Subsumption



In this scenario, the $\exists$-rule can be applied either first to $(x : \exists\, r\,.\, A)$ or to $(x : \exists\, r\,.\, A \sqcap B)$, because the order of rule application is not important to check satisfiability. However, the order of application results in two different completion graphs (see also figure 5.5) If $(\exists\, r\,.\, A \sqcap B)$ is expanded first, we get

$$\begin{aligned} G_{s2\exists,1} = &\ (\ V_{2\exists,1}, \quad E_{2\exists,1}, \quad \mathcal{L}_{2\exists,1}\ ) \\ = &\ \left(\{x, y\}, \quad \{r(x, y)\}, \quad \left\{ \begin{array}{l} (x : \exists\, r\,.\, A)\,, (x : \exists\, r\,.\, A \sqcap B)\,, \\ (y : A \sqcap B)\,, (y : A)\,, (y : B) \end{array} \right\}\right) \end{aligned} \tag{5.23}$$

with a single r-successor y. However, if $(\exists\, r\,.\, A)$ is expanded first, we get two r-successors y and z:

$$\begin{aligned} G_{s2\exists,2} = &\ (\ V_{2\exists,2}, \quad E_{2\exists,2}, \quad \mathcal{L}_{2\exists,2}\ ) \\ = &\ \left(\{x, y, z\}, \quad \{r(x, y), r(x, z)\}, \quad \left\{ \begin{array}{l} (x : \exists\, r\,.\, A)\,, (x : \exists\, r\,.\, A \sqcap B)\,, \\ (y : A \sqcap B)\,, (y : A)\,, (y : B)\,, \\ (z : A) \end{array} \right\}\right) \end{aligned} \tag{5.24}$$

Because $A \sqcap B$ subsumes A, expanding the former first causes the preconditions for $(x : \exists\, r\,.\, A)$ to become invalidated so $(x : \exists\, r\,.\, A)$ never gets expanded. Since $G_{s2\exists,1}$ and $G_{s2\exists,2}$ also have different representations as conjunctive queries, this situation is clearly not desirable.

It is possible to solve the problem by demanding that existential restrictions that refer to a subconcept (namely $\exists\, r\,.\, A$) are expanded before existential restrictions that refer to superconcepts (e.g. $\exists\, r\,.\, A \sqcap B$). Since the relationship between the subconcept and its superconcept may not be as easily discernible as in the example, the test would require a recursive reasoner invocation whenever an existential restriction is considered. This is not feasible performance-wise and another solution is highly desirable.

The same kind of non-determinism can also be provoked, if role hierarchies are present within the RBox. In the completion graph

$$\begin{aligned} G_{\sqsubseteq\exists} &= (\, V_{\sqsubseteq\exists}, \quad E_{\sqsubseteq\exists}, \quad \mathcal{L}_{\sqsubseteq\exists} \qquad\qquad\qquad\qquad\quad) \\ &= \Big(\{x\}\,, \quad \emptyset, \qquad \{(x : \exists\, r\,.\, A)\,, (x : \exists\, q\,.\, A)\}\Big) \end{aligned} \tag{5.25}$$

together with the RBox

$$\mathsf{RBox}_{\sqsubseteq\exists} = \{q \sqsubseteq r\} \tag{5.26}$$

tableau completion is once again non-deterministic. Depending on whether the $\exists$ rule first picks $(x : \exists\, r\,.\, A)$ or $(x : \exists\, q\,.\, A)$, we get two different saturated graphs.

If $(x : \exists\, q\,.\, A)$ is expanded first, we obtain

$$\begin{aligned} G_{s\sqsubseteq\exists,q} = &\ (\, V_{s\sqsubseteq\exists,q}, \quad E_{s\sqsubseteq\exists,q} \qquad \mathcal{L}_{s\sqsubseteq\exists,q} \qquad\qquad\qquad\qquad\quad) \\ = &\ \Big(\{x, y\}\,, \quad \{q(x, y)\}\,, \quad \{(x : \exists\, r\,.\, A)\,, (x : \exists\, q\,.\, A)\,, (y : A)\}\Big) \end{aligned} \tag{5.27}$$

Figure 5.6. Non-determinism during Tableau Expansion with Role Inheritance

(a) $G_{s \sqsubseteq \exists, q}$ **(b)** $G_{s \sqsubseteq \exists, r}$

Because $q \in r\downarrow$ from $\mathsf{RBox}_{\sqsubseteq\exists}$, this prevents generation of a second node for $(x : \exists\, q\,.\, A)$. However, when $(x : \exists\, r\,.\, A)$ is expanded first, we get

$$\begin{aligned} G_{s \sqsubseteq \exists, r} = &\ (\ V_{\sqsubseteq\exists,r}, & E_{\sqsubseteq\exists,r}, & & \mathcal{L}_{\sqsubseteq\exists,r} & \ ) \\ = &\ \left(\{x, y, z\}, \right. & \{r(x,y), q(x,z)\}, & & \left\{ \begin{array}{c} (x : \exists\, r\,.\, A), (x : \exists\, q\,.\, A), \\ (y : A), (z : A) \end{array} \right\} & \left. \right) \end{aligned} \tag{5.28}$$

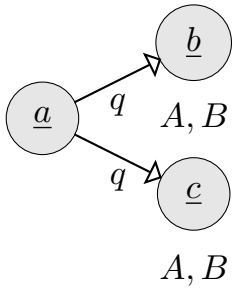
For $G_{s \sqsubseteq \exists, r}$, the concept description $(x : \exists\, q\,.\, A)$ is expanded first. Because having an r-successor does not imply a q-successor, expansion of $(x : \exists\, r\,.\, A)$ is performed afterwards, generating a new successor z.

The corresponding queries are

$$Q_{\sqsubseteq\exists,q} = \top(?x), q(?x, ?y), A(?x) \tag{5.29}$$

$$Q_{\sqsubseteq\exists,r} = \top(?x), r(?x, ?y), A(?x), q(?x, ?z), A(?z) \tag{5.30}$$

Figure 5.7. $KB_{2\sqsubseteq\exists}$: Query Coverage over Knowledge Bases



A closer look shows us that the interpretations of both queries are indeed the same. This has to be the case since otherwise the ∃-rule from the tableau would be faulty as it would change interpretations. However, $Q_{\sqsubseteq\exists,r}$ clearly differs from $Q_{\sqsubseteq\exists,q}$ not only syntactically. To illustrate this, consider the knowledge base (figure 5.7)

$$KB_{2\sqsubseteq\exists} = \{\top(\underline{a}), q(\underline{a},\underline{b}), q(\underline{a},\underline{c}), A(\underline{b}), B(\underline{b}), A(\underline{c}), B(\underline{c})\} \tag{5.31}$$

With $Q_{\sqsubseteq\exists,q}$, we need to assign either $\{\underline{a},\underline{b}\}$ or $\{\underline{a},\underline{c}\}$ to the query variables $\{?x,?y\}$. If $Q_{\sqsubseteq\exists,r}$ is chosen, instead, it is possible to cover both alternatives and an additional third one (with an additional symmetric). For the variables $\{?x,?y,?z\}$ the assignments $\{\underline{a},\underline{b},\underline{b}\}$, $\{\underline{a},\underline{c},\underline{c}\}$, and $\{\underline{a},\underline{b},\underline{c}\}$, $\{\underline{a},\underline{c},\underline{b}\}$ are all possible and are also proper interpretations of the initial completion graph.

The implemented solution is to this change the precondition of the ∃-rule in LillyTab's tableau (tables 4.3 and 4.4). Instead of demanding that a suitable successor with the qualified concept does not exist, the precondition is changed so that the ∃-rule can be invoked only once for each existential restriction term. This gives preference to larger (e.g. figure 5.5b and 5.6b) completion graphs, however it makes the ∃-rule deterministic without resolving to an expensive recursive reasoner invocation. The modified rule is easily implemented via the existing dependency tracking mechanism.

5.1.7. Benefits of Completion Graph Representation

In this chapter, a relationship between conjunctive queries and description logic completion graphs has been established. The mapping DL tableau allows for generation of conjunctive queries from many completion graphs as well as generation of completion graphs from all conjunctive queries.

The benefit of this duality is the fact that DL completion graph make exactly that information explicit that is only contained implicitly within a conjunctive query. Referring back to our example from section 1.2, we can identify some advantages:

1. In the source ontology (figure 1.3), the tableau will yield four distinct models for the different values (plaster, brick, lead, other) that are allowed to appear for the usesMaterial data property. We can use this as an indicator that distinct rules are required to map all cases.
2. In the target ontology, (figure 1.4), the fact that GroutFilling is a measurement becomes apparent because of the concept inheritance. Since we perform a full reasoning run, we capture all logical consequences. Existing ontology matchers heuristically capture only a few constraints, e.g. direct subclass relations.
3. In the target ontology, the fact that filledWith is a subproperty of usesMaterial will be readily apparent from the tableau. This gives us exactly the important piece of information to align filledWith with usesMaterial.
4. When (manually) introducing one of the subconcepts Plaster, Slate, Brick or Lead, we immediately gain the disjointness assertions.

5.2 A Mapping Tableau

While the basic tableau algorithm is already suitable to transform many completion graphs into a "mappable" saturated form, we have seen that it is insufficient in some cases and may even generate non-deterministic results with non-empty RBoxes. To facilitate extraction of conjunctive queries from DL completion graphs, a modified tableau is therefore required.

The new *mapping tableau* operates on a modified completion graph structure, that is augmented with three additional sets κ, the *governing terms,* Dep the *dependency map* and Mer the *merge map*.

In κ we keep track of those axioms that are important to the representation of the whole graph. Any representation created for the completion graph can start with the governing terms. Colloquially, the governing terms (together with some related axioms) contain all the information needed to represent a completion graph.

Dep keeps track of the reasoning process by recording the dependencies between concept axioms inserted into the completion graph.

For example, when $\{A \sqcap B\}$ is unfolded to $\{A \sqcap B, A, B\}$, we keep track of the fact that A and B were generated from $A \sqcap B$ in the dependency map.

Mer keeps track of node merges performed during tableau operation.

Because there is no room for confusion, as the augmented graph simply extends the basic completion graph with a additional sets, the same letters G and $\mathbb{G}$ will be used to represent augmented completion graphs.

Because of the new sets κ, Dep and Mer, a new merge-function is also required. The same translation that is required for $\mathcal{L}$ –rewriting all terms that refer to the source to refer to the target node of the merge– needs to be performed for κ and Dep. augmerge (function 5.2) performs the respective merge adoptions for κ, Dep, and Mer.

Tables 5.1 and 5.2 show the formal definition of the modified tableau. Consistency checking conditions are the same as with the original tableau. In comparison to the original tableau (tables 4.3 and 4.4), most of the rules have been modified only slightly. Tableau completion is started with κ initialized to $\mathcal{L}$. Dep and Mer are initially empty.

$\sqcap$ and $\sqcup$ The $\sqcap$- and $\sqcup$-rules are extended to propagate the governing term to their sub-axioms and are otherwise unchanged. The $\sqcup$-rule unconditionally creates new governing terms in the branches (G_{k+1}, G_{k+2}) because the introduced axioms are needed to differentiate the individual branches.

Definition 5.3 Augmented DL Completion Graph

An **augmented DL completion graph** over a logical language L (with signature Σ_L) is a 6-tuple $G \equiv_{\text{def}} (V, E, \mathcal{L}, \kappa, D, M)$ with

V a set of nodes $V \subset N$ forming the vertices of a graph, with N a totally ordered, countable, infinite set of node identifiers.

E a set of edges labelled with role names from $\mathcal{R}$, $E \subseteq \mathcal{R} \times V \times V$.

We write $r(x, y) \in E$, if $(r, x, y) \in E$.

$\mathcal{L}$ a set of concept labels, $\mathcal{L} \subseteq V \times \Sigma_L$.

We write $(x : C) \in \mathcal{L}$ if $(x, C) \in \mathcal{L}$ and $\mathcal{L}[x] \equiv_{\text{def}} \{C | (x : C) \in \mathcal{L}\}$.

κ a set of concept labels. $\kappa \subseteq \mathcal{L}$

Once again, we write $(x : C) \in \kappa$ if $(x, C) \in \kappa$ and $\kappa[x] \equiv_{\text{def}} \{C | (x : C) \in \kappa\}$.

If $(x : C) \in \kappa$, we call $(x : C)$ a *governing term*.

Dep the dependency map. $D \subseteq \mathcal{L} \times \mathcal{L}$.

We write $(x : A \to y : B) \in \text{Dep}$ if $(x, A, y, B) \in \text{Dep}$. $(x : A \to y : B) \in \text{Dep}$ indicates that the concept B at the node y was introduced as a logical consequence of the concept A at the node x.

Mer the merge map. $\text{Mer} \subseteq N \times N$.

If $(x, y) \in \text{Mer}$, x was merged into y.

Function 5.2: augmerge(G, t, s)

Input: $G = (V, E, \mathcal{L}, \kappa, D, M)$ an augmented DL completion graph
$t \in E$ the target node of the merge
$s \in E$ the source node of the merge

Output: $G_m = (V_m, E_m, \mathcal{L}_m, \kappa_m, \mathsf{Dep}_m, \mathsf{Mer}_m)$

begin

1 **if** *t is anonymous and s is not anonymous* **then**

2 $G_m \leftarrow \mathsf{augmerge}(G, s, t)$;

else

3 $V_m \leftarrow V - \{s\}$;

`/* Merge successor and predecessor links */`

4 $E_m \leftarrow (E - \{r(x,y) | x = s \vee y = s\})$
$\cup \{r(x,t) | r(x,s) \in E\}$
$\cup \{r(t,y) | r(s,y) \in E\}$;

`/* Merge node axiom sets to target */`

5 $\mathcal{L}_m \leftarrow (\mathcal{L} - \{(s:C) \,|\, (s:C) \in \mathcal{L}\})$
$\cup \{(t:C) \,|\, (s:C) \in \mathcal{L}\}$;

`/* Merge governing terms to target */`

6 $\kappa_m \leftarrow (\kappa - \{(s:C) \,|\, (s:C) \in \kappa\})$
$\cup \{(t:C) \,|\, C \in \kappa[s]\}$;

`/* Merge source dependency entries to target */`

7 $\mathsf{Dep}_m \leftarrow (\mathsf{Dep} - \{(x:C) \,|\, x = s \vee x = t\})$
$\cup \{(t:C \to y:D) \,|\, y \neq s \wedge (s:C \to y:D) \in \mathsf{Dep}\}$
$\cup \{(x:C \to t:D) \,|\, x \neq s \wedge (x:C \to s:D) \in \mathsf{Dep}\}$
$\cup \{(t:C \to t:D) \,|\, (s:C \to s:D) \in \mathsf{Dep}\}$;

`/* Record merge operation */`

8 $\mathsf{Mer}_m \leftarrow \mathsf{Mer} \cup \{(s,t)\}$;

9 $G_m \leftarrow (V_m, E_m, \mathcal{L}_m, \kappa_m, \mathsf{Dep}_m, \mathsf{Mer}_m)$;

end

10 **return** G_m;

end

$\forall$ The $\forall$-rule propagates governing terms into successors only if the supplied role is functional and a successor already exists. Otherwise the $\forall$-axiom is kept as a governing term, unmodified.

$\exists$ The $\exists$-rule is modified so that each existential quantification axiom is expanded exactly once.

This automatically implies termination of the algorithm, because the termination proof for blocking presented in [Tob01, p. 105] is still valid. There is only a finite number of $\exists$-axioms inside the completion graph and its TBox and no tableau operation removes axioms from an axiom set. Consequently, all $\exists$-axioms in a node will be expanded exactly once.

Soundness is guaranteed, because the modified rule always creates more successors than the original tableau. The number of $\exists$-invocations is strictly higher than that of the original tableau.

$\mathcal{F}$ and o The $\mathcal{F}$ and o rules have not been modified from their initial version. Tracking of governing terms κ for both rules is performed by the modified augmerge function

5.2.1. Extracting the Conjunctive Query

The tableau starts with an initial completion graph

$$G_0 = (V_0, E_0, \mathcal{L}_0, \kappa_0, \mathsf{Dep}_0, \mathsf{Mer}_0)$$

After tableau completion, we get a set of saturated completion graphs $\mathbb{G}_s$. For simplicity, we assume that there is only a single saturated completion graph $G_s \in \mathbb{G}_s$. If there are multiple completion graphs, the algorithm needs to be applied to every single saturated graph.

Table 5.1. Tableau Rules for the Mapping Tableau, part 1

Given a set $\mathbb{G} = \{G_1, \ldots, G_m\}$ of augmented DL completion graphs, for each graph $G_k \in \mathbb{G}$ and each node $x \in G_k$, repeat the following rules until no rule is applicable any more:

⊓-rule **if** 1. x is not blocked
2. $\exists\, C, D \,.\, (x : C \sqcap D) \in \mathcal{L}_k$
3. not $\{(x : C)\,, (x : D)\} \subseteq \mathcal{L}_k$

then update $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(x : C)\,, (x : D)\}$,
update $\mathsf{Dep}_k \leftarrow \mathsf{Dep}_k \cup \{(x : C \sqcap D \rightarrow x : C)\,, (x : C \sqcap D \rightarrow x : D)\}$.
if $(x : C \sqcap D) \in \kappa_k$,
then update $\kappa_k \leftarrow (\kappa_k - \{(x : C \sqcap D)\})$
$\kappa_k \leftarrow \kappa_k \{(x : C)\,, (x : D)\}$.

⊔-rule **if** 1. x is not blocked
2. $\exists\, C, D \,.\, (x : C \sqcup D) \in \mathcal{L}_k$,
3. $\{(x : C)\,, (x : D)\} \cap \mathcal{L}_k = \emptyset$,

then create two fresh successor graphs G_{k+1}, G_{k+2} and set

$G_{k+1} \leftarrow G_k$ $\qquad$ $G_{k+2} \leftarrow G_k$

$\mathcal{L}_{k+1} \leftarrow \mathcal{L}_k \cup \{(x : C)\}$ $\qquad$ $\mathcal{L}_{k+2} \leftarrow \mathcal{L}_k \cup \{(x : D)\}$

$\kappa_{k+1} \leftarrow (\kappa_k - \{(x : C \sqcup D)\})$ $\qquad$ $\kappa_{k+2} \leftarrow (\kappa_k - \{(x : C \sqcup D)\})$
$\kappa_{k+1} \leftarrow (\kappa_k \cup \{(x : C)\})$ $\qquad$ $\kappa_{k+2} \leftarrow (\kappa_k \cup \{(x : D)\})$

$$\mathbb{G} \leftarrow (\mathbb{G} - G_k) \cup \{G_{k+1}, G_{k+2}\}.$$

∀-rule **if** 1. x is not blocked,
2. $\exists\, r, C \,.\, (x : \forall\, r \,.\, C) \in \mathcal{L}_k$,

then for all $y \in V_k, q \in r{\downarrow}$ with $q(x, y) \in E_k$
update $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(y : C)\}$,
update $\mathsf{Dep}_k \leftarrow \mathsf{Dep}_k \cup \{(x : \forall\, r \,.\, C \rightarrow y : C)\}$,
if 1. $\mathsf{functional}(r)$,
2. $(x : \forall\, r \,.\, C) \in \kappa_k$,
3. $\exists\, y \in V_k, q \in r{\downarrow} \,.\, q(x, y) \in E_k$,
then $\kappa_k \leftarrow (\kappa_k - \{(x : \forall\, r \,.\, C)\}) \cup \{(y : C)\}$.

∀⁺-rule **if** 1. x is not blocked,
2. $\mathsf{transitive}(r)$,
3. $\exists\, r, C \,.\, (x : \forall\, r \,.\, C) \in \mathcal{L}_k$,

then for all $y \in V_k, q \in r{\downarrow}$ with $q(x, y) \in E_k$
update $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(y : \forall\, r \,.\, C)\}$,
update $\mathsf{Dep}_k \leftarrow \mathsf{Dep}_k \cup \{(x : \forall\, r \,.\, C \rightarrow \forall\, r \,.\, C)\}$.

Table 5.2. Tableau Rules for the Mapping Tableau, part 2

Continued from table 5.1.

Given a set $\mathbb{G} = \{G_1, \ldots, G_m\}$ of augmented DL completion graphs, for each graph $G_k \in \mathbb{G}$ and each node $x \in G_k$, repeat the following rules until no rule is applicable any more:

$\exists$-rule **if** 1. x is not blocked,
2. $(x : \exists\, r\,.\, C) \in \mathcal{L}_k$,
3. $\nexists y \in V_k\,.\, r(x, y) \in E_k \wedge (x : \exists\, r\,.\, C \rightarrow y : C) \in \mathsf{Dep}$,

then if 1. $\mathsf{functional}(r)$,
2. $\exists\, y \in V_k, q \in r{\downarrow}\,.\, q(x, y) \in E_k$,

then set $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(y : C)\}$,

else obtain a fresh, anonymous node $y \in N$,
set $V_k \leftarrow V_k \cup \{y\}$,
$E_k \leftarrow E_k \cup \{r(x, y)\}$, and
$\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(y : C)\}$

update $\mathsf{Dep}_k \leftarrow \mathsf{Dep}_k \cup \{(x : \exists\, r\,.\, C \rightarrow y : C)\}$.

if $(x : \exists\, r\,.\, C) \in \kappa_k$,

then update $\kappa_k \leftarrow (\kappa_k - \{(x : \exists\, r\,.\, C)\}) \cup \{(y : C)\}$.

$\mathcal{F}$-rule **if** 1. x is not blocked,
2. $\mathsf{functional}(r)$,
3. $\exists\, y, z\,.\, \{r(x, y), r(x, z)\} \subseteq E_k$,

then $G_k \leftarrow \mathsf{augmerge}\,(G_k, z, y)$.

***o*-rule** **if** 1. x is not blocked,
2. $\exists\, \underline{a} \in \mathcal{I}\,.\, \{\underline{a}\} \in \mathcal{L}_k[x]$,

then $G_k \leftarrow \mathsf{augmerge}\,(G_k, x, \underline{a})$.

If the governing term set κ of a saturated tableau graph generated using the mapping tableau contains only references to atomic concepts, it can be converted using a relatively simple algorithm. Fundamental to the algorithm is the fact that a node in the saturated graph is either 1. a *core node*, that was already part of the initial graph G_0, 2. was generated by some $\exists$-axiom.

Definition 5.4 Generating Existential Dependency

Given an augmented completion graph $G = (V, E, \mathcal{L}, \kappa, \mathsf{Dep}, \mathsf{Mer})$, a dependency map entry of the form

$$\exists\, y\,.\,(x : \exists\, r\,.\,C \rightarrow y : C) \in \mathsf{Dep} \wedge r(x, y) \in E$$

is called a **generating existential dependency**.
The dependency indicates that the node y was generated by the application of the $\exists$-rule from x.

Consequently, each node in the saturated completion graph is either a core node or there is a sequence of $\exists$-expansions that lead back to a core node. There are some particularities when core nodes are merged with other nodes, but the basic principle remains valid even then. The extraction algorithm can make use of the fact that every generated node can be traced back to a node that already existed in the initial graph. With this information, it is already possible to sketch the basic transformation algorithm:

1. For each node x that also appears in the initial graph G_0
   - if $\mathcal{L}_x$ contains a governing term, i.e. $\kappa[x] \neq \emptyset$, represent x using the governing terms,
   - otherwise represent x as $\top(?x)$ (using an appropriate query node mapping σ).
2. Represent all role links already present in the initial graph G_0.
3. Pick one of the nodes from the saturated graph G_s that contains at least one governing term. This includes nodes that where already present in the initial graph.
4. Follow **all** paths across existential generations ($(x : \exists\, r\,.\,C \rightarrow y : C) \in \mathsf{Dep}_s$) until the path reaches a node already covered.

5. Represent all nodes on the path and all connections on the path in the query. Roles are represented as binary predicates $r(?x, ?y)$ and nodes are represented using their governing terms or using $\top(?y)$.

6. Repeat until all governing terms have been consumed.

It is best to show the principal operation by a representative example. Assume that we want to convert the concept axiom

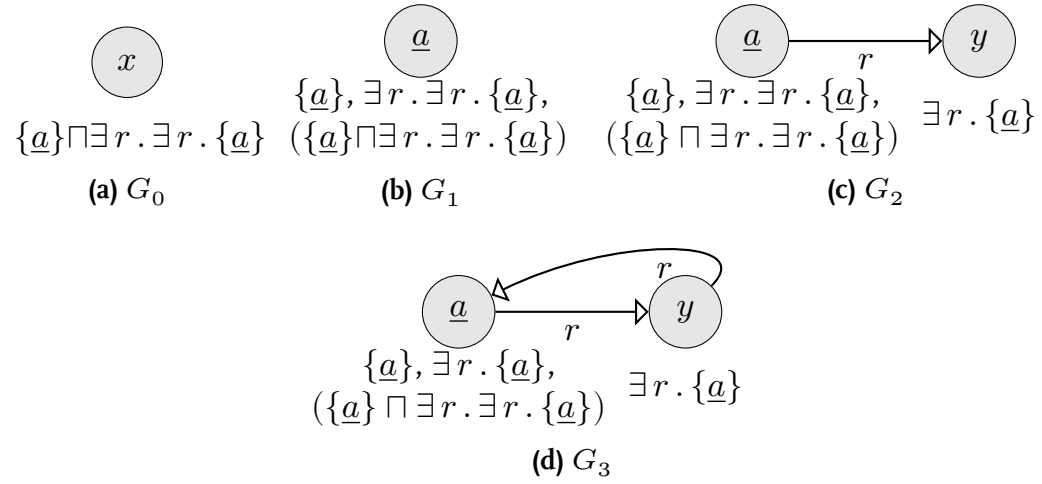
$$(\{\underline{a}\} \sqcap \exists r \,.\, \exists r \,.\, \{\underline{a}\}) \tag{5.32}$$

into a conjunctive query. The initial completion graph for this concept description is (see also figure 5.8a):

$$G_0 = \left(\begin{array}{ll} V_0 & = \{x\}\,, \\ E_0 & = \emptyset, \\ \mathcal{L}_0 & = \{(x : (\{\underline{a}\} \sqcap \exists r \,.\, \exists r \,.\, \{\underline{a}\}))\}\,, \\ \kappa_0 & = \{(x : (\{\underline{a}\} \sqcap \exists r \,.\, \exists r \,.\, \{\underline{a}\}))\}\,, \\ \mathsf{Dep}_0 & = \emptyset, \\ \mathsf{Mer}_0 & = \emptyset \end{array} \right)$$

Note that we use a slightly different notation here than in section 5.1, mixing the equality sign into the definition of the graph elements. While not formally correct, this change makes for improved reading for the six-tuple of the augmented graph.

Figure 5.8. Query Extraction using the Mapping Tableau



(a) G_0 **(b)** G_1 **(c)** G_2 **(d)** G_3

After processing of the $\sqcap$-rule, the graph G_1 (figure 5.8b) is produced by the tableau:

$$G_1 = \left(\begin{array}{ll} V_1 & = \{\underline{a}\}, \\ E_1 & = \emptyset, \\ \mathcal{L}_1 & = \{(\underline{a} : \{\underline{a}\}), (\underline{a} : \exists r . \exists r . \{\underline{a}\}), (\underline{a} : \{\underline{a}\} \sqcap \exists r . \exists r . \{\underline{a}\})\}, \\ \kappa_1 & = \{(\underline{a} : \{\underline{a}\}), (\underline{a} : \exists r . \exists r . \{\underline{a}\})\}, \\ \mathsf{Dep}_1 & = \left\{ \begin{array}{l} (\underline{a} : (\{\underline{a}\} \sqcap \exists r . \exists r . \{\underline{a}\}) \rightarrow \underline{a} : \{\underline{a}\}), \\ (\underline{a} : (\{\underline{a}\} \sqcap \exists r . \exists r . \{\underline{a}\}) \rightarrow \underline{a} : \exists r . \exists r . \{\underline{a}\}) \end{array} \right\} \\ \mathsf{Mer}_1 & = \{(x, \underline{a})\} \end{array} \right)$$

x has been merged as a consequence of the o-rule and is now represented by the distinct individual $\underline{a}$. Note, that the governing term set has changed and the governing terms are now the expanded sub-axioms $(\{\underline{a}\})$ and $(\exists r . \exists r . \{\underline{a}\}))$. The dependency map Dep contains a record of the performed transformations: the initial concept is recorded as the parent of both sub-axioms. Since entries in the dependency map Dep and the concept axiom set $\mathcal{L}$ are rather large, representation of Dep and $\mathcal{L}$ will be limited to the relevant axioms to reduce visual clutter.

In the next step, the tableau evaluates the axiom $(\underline{a} : \exists r . \exists r . \{\underline{a}\})$ and we obtain G_2 (fig. 5.8c):

$$G_2 = \left(\begin{array}{ll} V_2 & = \{\underline{a}, y\}\,, \\ E_2 & = \{r(\underline{a}, y)\}\,, \\ \mathcal{L}_2 & = \{\ldots, (y : \exists\, r\,.\,\{\underline{a}\})\}\,, \\ \kappa_2 & = \{(\{\underline{a}\} : \{\underline{a}\})\,, (y : \exists\, r\,.\,\{\underline{a}\})\}\,, \\ \mathsf{Dep}_2 & = \{\ldots, (\{\underline{a}\} : \exists\, r\,.\,\exists\, r\,.\,\{\underline{a}\} \rightarrow y : \exists\, r\,.\,\{\underline{a}\})\}\,, \\ \mathsf{Mer}_2 & = \{(x, \underline{a})\} \end{array} \right)$$

When y is generated, the governing term $(\{\underline{a}\} : \exists\, r\,.\,\exists\, r\,.\,\{\underline{a}\})$ is removed from $\underline{a}$ (previously x) and $(y : \exists\, r\,.\,\{\underline{a}\})$ is added to κ. The governing term is thus simplified and moves from $\underline{a}$ to y.

As a final tableau step, the axiom $(y : \exists\, r\,.\,\{\underline{a}\})$ is expanded. This results in an r-successor z of y which is automatically merged with $\underline{a}$ because of the nominal reference $\{\underline{a}\}$. The resulting graph is (see also figure 5.8d):

$$G_3 = \left(\begin{array}{ll} V_3 & = \{\underline{a}, y\}\,, \\ E_3 & = \{r(\underline{a}, y), r(y, \underline{a})\}\,, \\ \mathcal{L}_3 & = \mathcal{L}_2, \\ \kappa_3 & = \{(\{\underline{a}\} : \{\underline{a}\})\}\,, \\ \mathsf{Dep}_3 & = \{\ldots, (y : \exists\, r\,.\,\{\underline{a}\} \rightarrow \underline{a} : \{\underline{a}\})\}\,, \\ \mathsf{Mer}_3 & = \{(x, \underline{a})\} \end{array} \right)$$

In G_3 no tableau rules are applicable and there are no clashes, so G_3 is saturated (i.e. $G_3 = G_{s1}$. The governing term set κ_3 contains only a single, atomic concept axiom which indicates that it is indeed possible to convert G_3 into a conjunctive query.

The next step in the conversion algorithm is the representation of the all those nodes that were already present in the initial graph. Because the initial node x was merged into the named node $\underline{a}$, the set of nodes that are remaining from the initial graph is empty as x is not present any more. It is, however, clear that $\underline{a}$ must be represented in the query.

To solve this issue, it is possible to make use of the merge map. Tracking is provided by function mergeMap (function 5.3) that takes as input a graph node identifier x and returns the equivalent of x after node merging. If x was never merged, x is returned unmodified.

Function 5.3: mergeMap(G, x)

```
1 Input:  G = (V, E, L, κ, Dep, Mer)   an augmented completion graph
          x ∈ E                        a node in G
2 Result: x_m: the merge-mapped node corresponding to x or x, if x was never
          merged.
  begin
3   if ∃ y . (x, y) ∈ Mer then
4     x_m ← mergeMap(y);
    else
5     x_m ← x;
    end
6   return x_m;
  end
```

The core mapping algorithm only represents those nodes that are returned by the function mergeMap. For example, for G_3, mergeMap(G_3, x) returns $\underline{a}$. Representation of $\underline{a}$ is performed by a second function repr (function 5.4).

repr creates a representation of a node x in a graph G_s paying respect to the governing terms of x. If $\kappa_s[x]$ is empty, x's representation is simply $\top(?x)$. Otherwise x is represented using its governing terms: for each $A \in \kappa_s[x]$, x is represented as $A(?x)$. repr needs as input a query node mapping (see definition 5.2); for the example, we use $\sigma_q = \{\underline{a} \mapsto \underline{a}, y \mapsto ?y\}$. Because nominals are already handled by the query node mapping, governing terms referencing a nominal axiom can be safely ignored. In the example, repr represents $\underline{a}$ as $\top(\underline{a})$.

Function 5.4: repr(G, x, σ_q)

Input: $G = (V, E, \mathcal{L}, \kappa, \mathsf{Dep}, \mathsf{Mer})$ an augmented DL completion graph
x the node to represent
σ_q a query node mapping for G
Result: Q: A set of query axioms representing x in G.
initialise:;
1 $Q \leftarrow \emptyset$;
begin
2 **if** $\kappa[x] \neq \emptyset$ **then**
3 **foreach** $(x : C) \in \kappa[x] \wedge C \neq \{\underline{a}\}$ **do**
4 $Q \leftarrow Q \cup \{C(\sigma_q^{-1}[x])\}$;
end
else
5 $Q \leftarrow Q \cup \{\top(\sigma_q^{-1}[x])\}$;
end
6 **return** Q;
end

Conversion of the completion graph G_3 into a conjunctive query proceeds as sketched:

- $\underline{a}$ is mapped from x in the graph and $\kappa_3[x] = \{(x : \{\underline{a}\})\}$. Because $\{\underline{a}\}$ is a nominal reference, it is already represented by the query node mapping and not processed further.

  We represent x (from the original graph G_0) as $\top(\underline{a})$.

- Since there is only one core node and this core node has no role that refers to itself, no *core roles* are represented.

- However, $\underline{a}$ has a generating existential dependency $(y : \exists r \,.\, \{\underline{a}\} \rightarrow \underline{a} : \underline{a})$, which must be followed. The generating dependency points to y.

  As a consequence, $r(\underline{a}, ?y)$ is added to the conjunctive query.

- y does not appear in the initial graph and also has no governing terms, i.e. $\kappa_3[y] = \emptyset$. y is therefore represented using $\top(?y)$.

  y has a single generating existential dependency

  $(\{\underline{a}\} : \exists r \,.\, \exists r \,.\, \{\underline{a}\} \rightarrow \exists r \,.\, \{\underline{a}\} : \exists r \,.\, \{\underline{a}\})$.

The generating dependency points back to $\underline{a}$. Since x is present in the initial graph, no further tracking of $\exists$-expansion is required.

As a consequence of the generating dependency, $r(?y, \underline{a})$ is added to the conjunctive query.

- The resulting query is $\top(\underline{a}), r(\underline{a}, ?y), r(?y, \underline{a}), \top(?y)$.

  And this is indeed in line with the interpretation of the initial concept description $\{\underline{a}\} \sqcap \exists\, r\,.\,\exists\, r\,.\,\{\underline{a}\}$

The example has shown how nodes from saturated completion graphs are represented within conjunctive query terms. It has also been shown how tracking of generating existential dependencies can be used to generate a proper representation of many completion graphs as a conjunctive query.

The algorithm itself is split into two parts: the generation of the query terms for the core graph generateCoreQuery (function 5.5) and the resolution of the existential generating dependencies for all graph nodes with governing terms in function generateQuery (function 5.6).

Within generateCoreQuery X_0 represents the merged set of nodes from G_0 still present in G_s. If some $x_0 \in X_0$ was merged into another node, we track this merge (using, mergeMap, function 5.3). The loop in line 3 inserts the representations of core nodes into the result query Q. The loop in line 5 creates all role links required to represent the initial graph. The condition in line 6 is an optimization so that only the smallest roles for any link are represented. Role axioms where we have a subrole link between the same source and target node are not created.

Returned from generateCoreQuery is a set of query terms that map the nodes that where already present in the initial graph. After generating the core query using the already defined generateCoreQuery (function 5.5), the remaining nodes that have at least one governing term are considered.

Function 5.5: generateCoreQuery(G_0, G_s, σ_q)

Input: $G_0 = (V_0, E_0, \mathcal{L}_0, \kappa_0, \mathsf{Dep}_0, \mathsf{Mer}_s)$ An augmented DL completion graph

$G_s = (V_s, E_s, \mathcal{L}_s, \kappa_s, \mathsf{Dep}_s, \mathsf{Mer}_s)$ The saturated, augmented DL completion graph generated from G_0 using the mapping tableau

σ_q a query node mapping for all nodes in G_s

Result: Q: A conjunctive query

initialize:;

1 $Q \leftarrow \emptyset$;

begin

2 $X_0 \leftarrow \{\mathsf{mergeMap}(G_s, x) | x \in V_0\}$;

3 **foreach** $x_0 \in X_0$ **do**

4 $Q \leftarrow Q \cup \mathtt{repr}(G_0, x, \sigma_q)$;

end

5 **foreach** $r(x, y) \in E$ **do**

6 **if** $\{x, y\} \subseteq X_0 \wedge (\nexists q \,.\, q \neq r \wedge q \in r\downarrow \wedge q(x, y) \in E)$ **then**

7 $Q \leftarrow Q \cup \left\{r(\sigma_q^{-1}[x], \sigma_q^{-1}[y])\right\}$;

end

end

8 **return** Q;

end

To represent non-core axioms, generateQuery (function 5.6) continues processing. In generateQuery, the set Y is the *open list*, containing those nodes that still need to be processed. Y is initialized to the set of those nodes, that have at least one governing term (line 4). The algorithm picks a node $y \in Y$ and adds its representation to the query (line 10).

The algorithm builds a list of generating nodes (i.e. those that have an existential axiom that is expanded into y and adds role links from the generating node z to y (line 12). Because $\mathcal{SHOF}$ has the tree model property and G_s is therefore a forest with the root nodes in the core nodes X_0, generateQuery always terminates.

Function 5.6: generateQuery(G_0, G_s)

Input: $G_0 = (V_0, E_0, \mathcal{L}_0, \kappa_0, \mathsf{Dep}_0)$ an augmented completion graph
$G_s = (V_s, E_s, \mathcal{L}_s, \kappa_s, \mathsf{Dep}_s)$ one of the saturated, augmented completion graphs for G_0 generated using the mapping tableau

Result: Q: A conjunctive query

initialize:;

1 $\sigma_q \leftarrow$ a query node mapping for G_s;
2 $Q \leftarrow$`generateCoreQuery` (G_0, G_S, σ_q);
3 $X_0 \leftarrow \{\mathsf{mergeMap}(G_s, x) | x \in V_0\}$;
4 $Y \leftarrow \{y \in V_s | \kappa_s[y] \neq \emptyset\}$;
5 $Z \leftarrow V_s - X_0$;
begin
6 **while** $Y \neq \emptyset$ **do**
7 pick a node y from Y;
8 $Y \leftarrow Y - \{y\}$;
9 $Z \leftarrow Z - \{y\}$;
10 $Q \leftarrow Q \cup \mathtt{repr}(G_s, y, \sigma_q)$;
11 **foreach** $z \in \{z \in V | (z : \exists r \,.\, C \rightarrow y : C) \in \mathsf{Mer}_s\}$ **do**
12 $Q \leftarrow Q \cup r(\sigma_q^{-1}(z), \sigma_q^{-1}(y))$;
13 **if** $z \notin Z$ **then**
14 $Y \leftarrow Y \cup \{z\}$;
end
end
end

5.2.2. Query Complexity

Execution of a mapping requires both efficient querying of the source knowledge base and efficient creation of the target graph. At the condition side, evaluating general conjunctive queries is known to be NP-complete [CM77], query output (i.e. finding all consistent assignments to the free variables) for acyclic queries is LOGCFL-complete [GLS01]. Research on answering conjunctive queries over description logic knowledge bases is still ongoing, but decidable algorithms have been found for many expressive description logics ([Sch96, OCE08, CDGL$^+$13]). The upper bound for answering conjunctive queries over all sublogics of $\mathcal{SHOF}$ is in CoNP [OCE08]. Even though these are discouraging complexity results, queries generated for mappings are usually small and we assume that query answering over the source ontology is not a problem.

5.3 Tuple Generating Dependencies

Given the results from the previous section, it seems natural to extend the correspondence between model graphs and conjunctive queries also to mappings. When we look back to section 1.2 (the initial example), we can observe that the mapping there was obtained by specifying a partial ontology graph at both the source and the target side. The implicit assumption there was, that one instantiation of the target structure would be generated for each individual that matches the source model. This type of translation rule is commonly used for relational databases, where we can call it a *view* or –formally– a *tuple generating dependency* [Fag09].

Tuple generating dependencies capture the intended meaning of a directed mapping. The *condition* part of a tuple generating dependency is executed as a query on a source knowledge base. The *entailment* part is invoked for each result returned by the source query and generates the desired structures in the target ontology. In the literature, the terms tuple generating dependency and *rule* are often used interchangeably.

Definition 5.5 Tuple Generating Dependency

Let $\mathcal{I}$ be a set of individuals, $\mathcal{V}$ a set of variables, $X_f \subseteq \mathcal{V}$, $Y \subseteq \mathcal{V}$, $Z \subseteq X_f \cup Y$ be sets of variables, and P_i and Q_j be atomic predicates over individuals and variables. A **tuple generating dependency** (tgd) is an expression of the form

$$\begin{aligned} \text{tgd:} \quad & \forall\, X_f \,.\, \phi(X_f) && \mapsto \exists\, Y \,.\, \psi(Z) \\ = \quad & \forall\, X_f \,.\, \underbrace{\bigwedge_i P_i\,(X_i)}_{\text{condition}} && \mapsto \exists\, Y \,.\, \underbrace{\bigwedge_j Q_j\,(Z_j)}_{\text{entailment}} \end{aligned}$$

with $X_i \subseteq X_f \cup \mathcal{I}$ and $Z_j \subseteq Z \cup \mathcal{I}$ for every i and j.
The subterm $\phi = \bigwedge_i P_i\,(X_i)$ is called the **condition**, **body**, or **source side**. The subterm $\psi = \bigwedge_j (Z_j)$ is called the **entailment**, **head** or **target side** of the dependency. Because X_f and Y are clear from ϕ and ψ, the tuple generating dependency is usually written in short form $\phi \mapsto \psi$.

Of most interest for this work is the fact that most tuple generating dependencies can be expressed as a pair of DL completion graphs. This means, that instead of writing a query

$$\forall\, X \,.\, Q_s(X) \mapsto \exists\, Z \,.\, Q_t(X \cup Z) \tag{5.33}$$

it is instead possible to give two completion graphs G_s and G_t, which represent Q_s and Q_t, respectively. G_s is interpreted as a query of the source knowledge base. The anonymous nodes in G_s are treated as free variables of the query. If G_s and G_t share nodes, each set of individuals that match these common nodes is replaced into a separate instance of the target graph.

Definition 5.6 Mapped Node

Given a mapping rule $\beta = (G_s, G_t)$, expressed as a pair of completion graphs (G_s, G_t), x is a **mapped node**, if both $x \in V_s$ and $x \in V_t$ and x is also anonymous in both G_s and G_t.

The application of such a mapping loops over all sets of input individuals and creates appropriate structures on the target side as indicated by G_t. For example, given the completion graph pair G_s and G_t (see also figure 5.9)

$$G_s = \begin{pmatrix} V_s & = \{x, y\} \\ E_s & = \{r(x, y)\} \\ \mathcal{L}_s & = \{(x : A), (y : B)\} \end{pmatrix} \tag{5.34}$$

$$G_t = \begin{pmatrix} V_t & = \{x, y, z\} \\ E_t & = \{q(x, z), p(z, y)\} \\ \mathcal{L}_t & = \{(x : C), (z : D), (y : E)\} \end{pmatrix} \tag{5.35}$$

and the source knowledge base KB_s (left side of figure 5.10), the individuals $\underline{a}$ and $\underline{b}$ are matched by x in G_s and $\underline{c}$ and $\underline{d}$ are matched by y in G_s with the link matched by the asserted link between the respective individuals.

$$KB_s = \{A(\underline{a}), r(\underline{a}, \underline{b}), B(\underline{b}), A(\underline{c}), r(\underline{c}, \underline{d}), B(\underline{d})\} \tag{5.36}$$

Consequently, two result instances are returned from matching G_s against KB_s:

$$\begin{aligned} x \leftarrow \underline{a}, \quad y \leftarrow \underline{b} \\ x \leftarrow \underline{c}, \quad y \leftarrow \underline{d} \end{aligned} \tag{5.37}$$

For each of these results, an instance of G_t is created in the target knowledge base KB_t. x and y in G_t are replaced by the variable assignments obtained from the source graph. Because G_t also holds an additional, anonymous individual z, we generate fresh individuals for each instance of G_t, too. The process is depicted in figure 5.10.

Function primitiveApplyTGD (function 5.7) gives a simple algorithm for the execution of completion graph based mappings. The primitive algorithm, however, makes a few assumptions that need to be resolved before applying it as an algorithm for the execution of a completion graph based alignment.

Function 5.7: primitiveApplyTGD((ϕ, ψ), O_s, O_t)

Input: $(\phi, \psi) = \phi \mapsto \psi$ a tuple generating dependency
O_s a source ontology
O_t a target ontology

Output: O_t

$R \leftarrow$ the query results obtained from executing ϕ over O_s.

`/* Each` $\sigma \in R$ `is then a set of variable assignments of each of the variables` X_f `in` ϕ. `*/`

foreach $\sigma \in R$ **do**

Calculate $\psi[\sigma]$ by applying the assignments from R to ψ.

define a mapping function $m_{\psi[\sigma]}$ such that

$$m_{\psi[\sigma]}(x) = \begin{cases} \underline{a} & x = \underline{a} \in \mathcal{I} \\ \underline{y} & x = ?y \in Y \text{ with } \underline{y} \text{ a unique, unused individual name for each variable } ?y. \end{cases}$$

foreach $C(x) \in \psi[\sigma]$ *with* $x \in \mathcal{I} \cup Y$ **do**

$O_t \leftarrow O_t \cup \left\{C(m_{\psi[\sigma]}(x))\right\}$;

end

foreach $r(x, y) \in \psi[\sigma]$ *with* $x, y \in (\mathcal{I} \cup Y)$ **do**

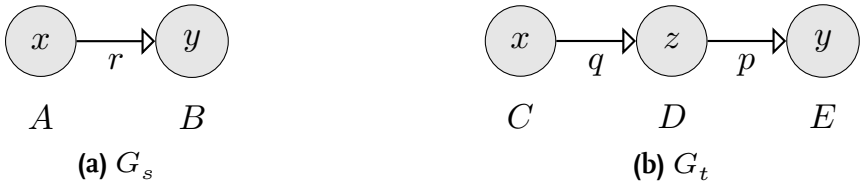
$O_t \leftarrow O_t \cup \left\{r(m_{\psi[\sigma]}(x), m_{\psi[\sigma]}(y))\right\}$;

end

return O_t;

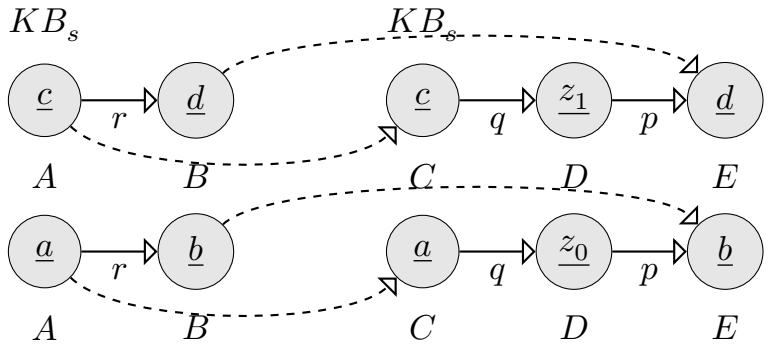
end

Figure 5.9. Example Mapping as Model Graphs



(a) G_s **(b)** G_t

Figure 5.10. Knowledge Bases as Source and Target in Mapping



Dashed gray lines (- -▷) indicate nodes transferred from KB_s to KB_t. $\underline{z_0}$ and $\underline{z_2}$ are fresh individuals, previously unused.

source-to-target property The procedure assumes that a single cycle of query execution is sufficient to generate all query results. This means, that we assume that the source schema is completely separate from the target schema and –consequently– the execution of query heads does not influence the source ontology.

consistency and coherence Possible conflicts generated due to invocation of multiple rules are not handled. Rule coherence (section 3.2.4) is not guaranteed.

5.3.1. Source-to-Target Property

A conjunctive query has the *source-to-target* property, if the source schema and the target schema are disjoint. Intuitively, this means that assertions generated during query execution of the head statements do not interfere with the satisfiability of the body conditions.

The *source-to-target* property has practical significance. Reasoning in $\mathcal{SHOIN}(\mathbf{D})$ extended with tuple generating dependencies is undecidable [MSS05]. Rule languages that can be embedded into an expressive DL ontology are therefore limited to the set of *DL-safe* rules. DL-safe rules is the name used by the semantic web community for what is known in relational database research as the set of *full tgds*. A tgd (definition 5.5) is "full", if $Y = \emptyset$, that is when the rule cannot generate new individuals but only add to existing individuals.

Full tgds avoid a problem of recursive expansion that we have already seen in a similar way in section 4.2.1. Given the tgd

$$\forall\, ?x\,.\, A(?x) \Rightarrow \exists\, ?y\,.\, r(?x, ?y), A(?y) \tag{5.38}$$

we can see that such a rule potentially creates individuals matching its own precondition. Consequently, another rule execution cycle is needed, because more individuals may now match the condition of the rule. There are many knowledge bases that can be generated from the above rule, and once again infinite expansion may only be prevented by suitable blocking. In fact, the tgd in equation (5.38) is equivalent to the TBox axiom $A \sqsubseteq \exists r\,.\,A$. Unfortunately, in the mapping scenario, not being able to generate new individuals in the target ontology, makes full tgds unsuitable as a mapping framework.

To solve this problem, it seems appropriate to have a look at the ontologies the mapping process is expected to handle in the case of document ontologies. If source (O_s) and target ontology (O_t) do not share a common signature, i.e. if $\mathcal{C}_s \cap \mathcal{C}_t = \{\top, \bot\}$ and $\mathcal{R}_s \cap \mathcal{R}_t = \emptyset$ the simple algorithm is applicable. However, if O_s (i.e. Σ_{L_s}) and O_t (Σ_{L_t}) share a set of common axioms, the situation is more difficult. When both ontologies share at least a common concept or role (or inherit a common role or concept from a common upper ontology) adding new axioms in one (source or target) ontology can change the meaning of concepts in the other ontology or in the upper ontology. This is clearly not a desirable situation.

If our source and our target ontology have common elements, we want to make sure that assertions in the target ontology do not affect the validity of concepts in the source ontology. The notion of not being able to change the meaning of

existing assertions by extending the existing ontology is captured by the formal definition of a *conservative extension*. If we start with an ontology O (the upper ontology), adding another ontology O_{ext}, the union $O \cup O_{\text{ext}}$ of both ontologies should not change the meaning of the elements from O.

Definition 5.7 Deductive Conservative Extension
Given two ontologies O, and O_{ext} (over language L)
O_{ext} is called a **deductive conservative extension** [LWW07] of O
iff $\forall\, \phi \in \Sigma_L \,.\, O \models \phi \Leftrightarrow O_{\text{ext}} \models \phi$.

The problem of conservative extensions has caught previous interest in the research for automated modularizations of ontologies. Determining, if an extension O_{ext} is a conservative extension of another ontology O is, unfortunately, 2EXPTIME-complete already in $\mathcal{ALCQI}$ [LWW07]. However, Grau et al [GHKS08] provide both a sufficient semantic and a sufficient syntactic condition to test for conservative extensions. The semantic condition involves concept checking using a DL reasoner. The syntactic condition is only defined over the syntax of concepts allowed in the TBox and RBox of the combined ontology. The original paper also contains a statistics of locality testing for existing ontologies and has found that only 11 of 400 tested ontologies violate the locality condition.

It seems therefore safe to assume that assertions made in the target ontology do not influence the source ontology. Consequently, because (in a rule $\phi \mapsto \psi$) all conditions in ϕ are defined over the source ontology and all entailments in ϕ are defined over the target ontology, a single cycle of rule execution is sufficient and additionally, the union of source ontology and the mapping ruleset remains decidable.

Chapter Summary

In this chapter, we have described a mapping between conjunctive queries and DL completion graphs. We have developed algorithms to convert one into the other and vice versa. Since the conversion is not always possible, we have highlighted the problem areas and provided solutions for many of the mapping problems.

We now have a technique to represent bridge rules (definition 3.19) in the form of tuple generating dependencies (definition 5.5) as pairs of DL completion graphs. That is, we can now write $\beta = (G_s, G_t)$ for many types of completion graphs. We have modified an existing description logic tableau to provide necessary information to aid in the conversion in both directions, i.e. from β to (G_s, G_t) and vice versa. To further establish the usefulness of the approach, we have confirmed that execution complexity of *model based* mapping does not seem to be a problem in practice.

SECTION 6.0

Mapping Refinement

The last chapter has introduced a method to represent bridge rules (in the form of tuple generating dependencies) as pairs of DL completion graphs.

In this chapter, we extend on those results to establish the concept of *mapping refinement*. Refinement is a rule-based, iterative process to derive complex alignments starting from simple correspondences.

We go on to describe *model based refinement* as a concrete variant of mapping refinement, using completion graphs as the primary representation mechanism. On top of this model based approach, semantic refinement rules are developed, yielding a concrete implementation of model based refinement.

Having developed a framework for the representation of complex mappings between ontologies, in the second step we return to the initial scenario. In the format translation use case, we have a set of simple correspondences $\mathbb{A}$. This simple alignment has been obtained by leveraging existing matcher technology and has already been reviewed. It also seems safe to assume, that the alignment is correct. We assume that the correspondences in the alignment are directional and thus have the *source-to-target* property (see also section 5.3.1). Simple, directional, source-to-target properties can be formulated as tuple generating dependencies. The corresponding dependencies are then of the form

$$\forall\, ?x\,.\, C_{s,i}(?x) \mapsto C_{t,i}(?x)$$

However, it is known, that the initial alignment is not complete and it misses some of the information that can be transferred from the source to the target ontology. This is only possible using complex mapping rules that do not only evaluate a single concept or role, but can evaluate (and generate) multiple, interlinked ontology elements.

In this scenario, it seems natural (as shown in section 1.2) to start of with the initial alignment and iteratively refine the already discovered correspondences into more complex alignment rules. A demonstration of the process has already been given in the introductory example in section 1.2. The example has also shown, that manual derivation of complex mappings from less complex mappings is a cumbersome and error-prone task, so that at least some kind of automation is highly desirable.

The problem at hand is similar to that of inductive logic programming (ILP) [Mug91]. In inductive logic programming, one is presented with a set of input examples and a set of predicates. The goal for an inductive logic learning system is then to find the best set of predicates (and variable allocations), such that the input examples are classified with the best possible precision. Stuckenschmidt et al [SPM08] have proposed to adapt ILP to the task of complex ontology matching.

However, the matching task differs from the ILP problem in some regards:

Refinement is not classification learning Traditional ILP attempts to learn a classification, i.e. a formula that classifies the input examples. It starts from a simple formula and iteratively adds new clauses to *refine* the classification. The quality of the new formula is checked in each step using the training examples.

For ontology matching, training examples need to be in the form of pre-matched instances, i.e. individuals that are asserted to belong to both the source and target ontology. ILP does not offer direct support to calculate these training examples. If no training examples are available –like in the example scenario from section 1.2– ILP cannot be applied.

Source and target of a correspondence are linked Traditional ILP learns a formula as a classification. While it is natural to look at ILP to learn both sides of a mapping rule as formulæ, the interaction between both sides causes a significant increase in complexity. Both sides cannot be learned independently and the refinement process must be controlled on both sides simultaneously.

Quality evaluation is different Traditional ILP assumes a high quality evaluation function (namely verifying against the training examples). If training examples are not available, only the similarity between both sides of a mapping can be used as an indicator. Designing a suitable similarity measure has been the focus for research in ontology matching for quite some years for simple alignments alone. Introducing similarity between complex mapping rules can be expected to be an order of magnitude more complex.

DAG-shaped hypothesis lattice Many primitive ILP systems assume that adding a predicate makes a formula more restrictive. However, this syntactic notion of refinement is only true when there is no background knowledge. In the face of a full ontology with formal semantics, the assumption no longer holds. A similar problem appears with regard to confluence. Given a two sequences of refinement actions $\mathrm{P} = (\rho_0, \ldots, \rho_k)$ and $\mathrm{P}' = (\rho'_0, \ldots, \rho'_m)$ that are sequentially applied to the same initial state s, let the results of performing these steps be $\hat{s} = \rho_0 \circ \ldots \circ \rho_k(s)$ and $\hat{s}' = \rho'_0 \circ \ldots \circ \rho'_m(s)$, respectively. $\hat{s}$ and $\hat{s}'$ need not share any syntactic features, but (when incorporating background knowledge) they still may be semantically equivalent, i.e. $\hat{s} \equiv \hat{s}'$, semantically. This result is possible, even if P and P' do not share any individual actions. Because of the formal semantics, the information fragment introduced by the refinement needs to be combined with the involved ontologies' TBoxes. Background knowledge thus needs to be considered when designing a refinement process for complex ontology alignment.

None of these concerns invalidate the basic principle of ILP, however. It still seems reasonable to start with an initial (simple) alignment as these are obtainable from existing matchers. By looking at the initial correspondences together with the associated ontologies, it is possible to determine *refinement actions* to iteratively augment the initial mappings

Applying the *refinement* approach to mappings expressed as DL completion graphs is particularly promising, as DL completion graphs form a natural, mostly syntactic *refinement hierarchy* that can be exploited for mapping.

Refinement as Extraction of Semantic Features At a high level, the refinement process can be described as an *enrichment*. Each refinement step adds another piece of information to an existing completion graph

This "piece of semantic information" concept has been mentioned before in section 2.6 and it seems obvious to make the connection: The intended semantics of a refinement step is that each step adds another semantic feature to the completion graph. This leaves room for domain-specific refinement rules and even sets of refinement patterns similar to the mapping patterns described in [Sch09].

6.1 Refinement Process

Starting from an initial correspondence, it is possible to define the abstract *refinement process* as a framework with well-defined inputs and outputs. This refinement process takes as input

$\boldsymbol{O_s, O_t}$ two ontologies the source (O_s) and target (O_t) ontology of an alignment.

$\boldsymbol{S_{\text{ref}}}$ a set of refinement states. These come in two variants, namely $S_{\text{ref},s} \subset S_{\text{ref}}$ for the source ontology and $S_{\text{ref},t} \subset S_{\text{ref}}$ for the target ontology.

There are two sets of refinement states, $S_{\text{ref},s}$ and $S_{\text{ref},t}$ with members $s_{\text{ref},s,i}$ and members $s_{\text{ref},t,j}$, respectively. These identifiers will be shortened to, S_s (S_t) and $s_{s,i}$ ($s_{t,j}$) when there seems to be no chance of confusion.

$\mathbb{A}$ an initial, simple alignment. As noted, this will be obtained using an existing (simple) matcher.

sim_S a similarity between refinement states S_{ref}.

term a relation, with term $\subseteq S_{\text{ref}} \times \Sigma_{L_{\text{conjunctive}}}$ correlating *refinement states* S_{ref} with conjunctive terms.

$\Sigma_{L_{\text{conjunctive}}}$ is the language of individual conjunctive terms, e.g. $P_i(X_i) \in \Sigma_{L_{\text{conjunctive}}}$.

P a (possibly infinite) set of refinement rules $\mathrm{P} \equiv_{\text{def}} \{\rho | \rho \mapsto S_{\text{ref}} \times \mathfrak{P}(S_{\text{ref}})\}$
Each refinement rule ρ maps a state to zero or more *successor states*.

If $\rho(s_{\text{ref}}) \neq \emptyset$ for some $s_{\text{ref}} \in S_{\text{ref}}$, ρ is said to be *applicable* in s_{ref}.

ref_s, ref_t Two conjunctive reference term functions

$$\mathsf{ref}_s : \mathbb{e}_s \mapsto \Sigma_{L^+_{\text{conjunctive}}}$$

$$\mathsf{ref}_t : \mathbb{e}_t \mapsto \Sigma_{L^+_{\text{conjunctive}}}$$

with $\mathbb{e}_s$ ($\mathbb{e}_t$) a subset of the source (target) elements referenced from $\mathbb{A}$:

$$\mathbb{e}_s \subseteq \{e_s | (e_s, e_t, n) \in \mathbb{A}\}$$

$$\mathbb{e}_t \subseteq \{e_t | (e_s, e_t, n) \in \mathbb{A}\}$$

$\Sigma_{L^+_{\text{conjunctive}}}$ is the language of conjunctive terms, i.e. intersections of conjunctive clauses, e.g. $P_i(X_i) \wedge P_j(X_j) \in \Sigma_{L^+_{\text{conjunctive}}}$.

ref_s and ref_t map elements from the source and target alignments to reference terms. These reference terms correspond to the user-supplied queries that should be answered by the refinement process. More details will be given in section 6.1.3.

Both ref_s and ref_t are optional. Typically, only one of them is supplied.

In the initial step, the abstract refinement process picks a correspondence $A \in \mathbb{A}$. From A it generates two initial states $s_{s,0}$ and $s_{t,0}$ representing the initial, simple alignment (not the reference terms).

The refinement process then searches for applicable rules ρ_s (ρ_t) from P. If a rule is applicable in $s_{s,i}$ ($s_{t,j}$), one or more *successors* are generated by evaluating $\rho(s_{s,i})$ ($\rho(s_{t,j})$) for each state until a termination condition (see section 6.1.3) is reached or until there are no more applicable refinement rules. The result of this process is two sets of refinement graphs (definition 6.1), $\mathbb{G}_{\mathrm{ref},s}$ and $\mathbb{G}_{\mathrm{ref},t}$, one for the source and one for the target side of the alignment.

A refinement graph is a directed, acyclic graph with refinement states as vertices and refinement rules as edges pointing to refined successor states. In many cases, refinement graphs will have the tree property, but logical derivates make it possible for the tree to degenerate into a DAG.

Definition 6.1 Refinement Graph

Let S_{ref} be a set of refinement states and P be a set of refinement rules. A refinement graph $\mathbb{G}_{\mathrm{ref}}$ is a labelled graph $\mathbb{G}_{\mathrm{ref}} \equiv_{\mathrm{def}} (S_{\mathrm{ref}}, E_{\mathrm{ref}}, \mathcal{L}_{\mathrm{ref}})$ with

- vertices $s_{\mathrm{ref}} \in S_{\mathrm{ref}}$ and
- edges $E_{\mathrm{ref}} \subseteq {S_{\mathrm{ref}}}^2$
- edge labels $\mathcal{L}_{\mathrm{ref}} : E_{\mathrm{ref}} \mapsto \mathrm{P}$, assigning some $\rho \in \mathrm{P}$ for each edge.

and the property that every edge must be the result of a refinement rule application:

$$\forall\, s_x \in S_{\mathrm{ref}} \,.\, \left\{ s_y \middle|\, (s_x, s_y) \in E_{\mathrm{ref}} \right\} \subseteq \bigcup_{\rho \in \mathrm{P}} \rho(s_x)$$

6.1.1. Model-Based Refinement

With the results from chapter 5, we can instantiate the abstract refinement process into a concrete version:

- States are represented by completion graphs, i.e. $s_{\mathrm{ref}} = G_{\mathrm{ref}}$ for some completion graph G_{ref}.
- term is implemented by the conjunctive query repr function (function 5.4). repr turns a completion graph into a conjunctive term as required.

- sim_S becomes $\mathsf{sim}_G : \mathbb{G} \mapsto \mathbb{R}$, a similarity function between DL completion graphs.

- Refinement rules P introduce new terms into a completion graph.

Figure 6.1 shows two model based-refinement trees for the introductory example (section 1.2). As we can see, the initial mapping is represented in the root nodes of both trees. Both root graphs contain a single node marked with the concepts in the initial GroutFilling $-$ GroutFilling correspondence. Introducing the term $(x : \exists\, \mathsf{uM}\,.\,\top)$ on the source side and running the reasoner results in multiple successor graphs. A similar process can be seen at the target side. Here, however two refinement steps have been performed.

6.1.2. Refinement Strategy

Looking back at the original scenario (section 1.2), we can state that the goal of migration is to extract certain information from a historic (source) ontology. Under the assumption that the user is familiar with the current (target) ontology, it seems reasonable for her to provide queries to be answered on the target ontology. It is only the results of these queries that need to be mapped.

These user-supplied queries are represented by ref_s (ref_t), respectively. For some input correspondence $c \in \mathbb{A}$, we get a set of *reference terms* that we want a refined alignment for.

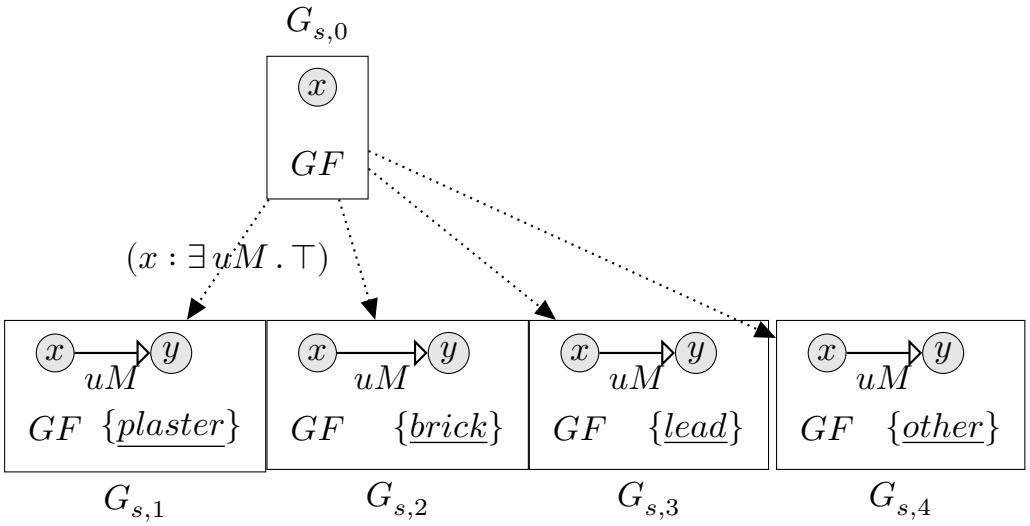
For the refinement process, ref_s (ref_t) serves two purposes:

termination critierion When a state s_{ref} is generated during refinement with

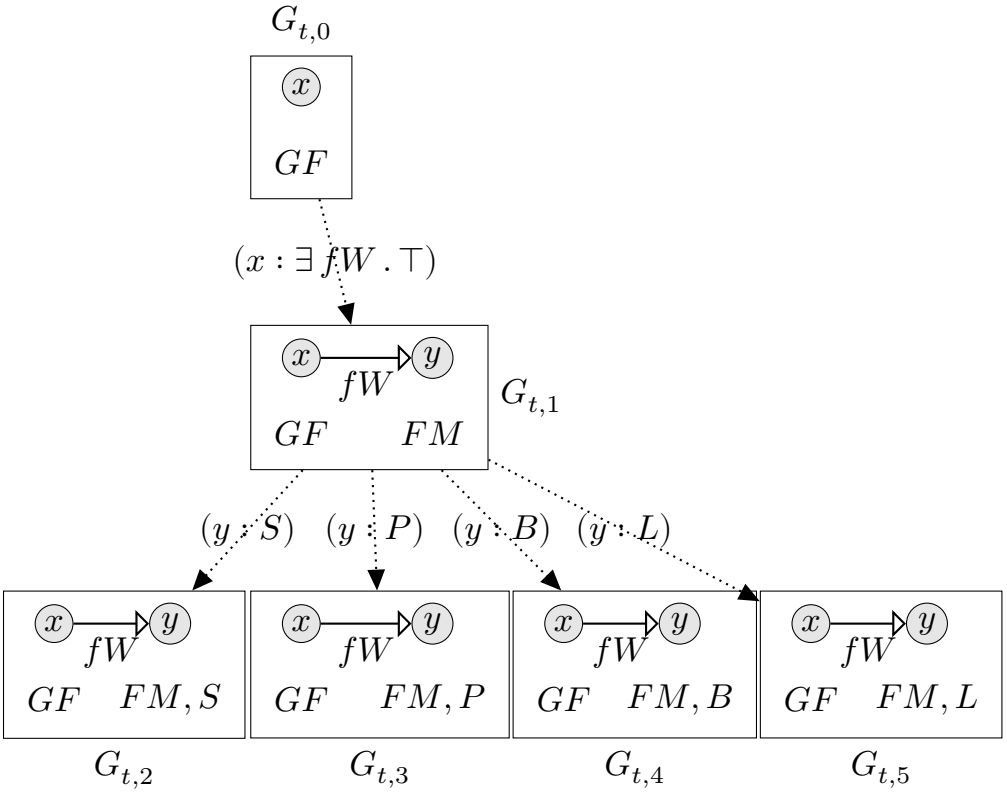
$\mathsf{term}(s_{\mathrm{ref},i}) \in \mathsf{term}_s(e)$, then no more successors need to be generated for $s_{\mathrm{ref},i}$.

Because we have already reached the point where the user's query is represented, no more refinement is necessary.

Figure 6.1. Example Refinement Graph Fragment



(a) Source



(b) Target

Shortened labels are taken from table 1.2.

state selection If ref_s (ref_t) is specified, it also influences the selection of refinement states that are considered for a mapping. Because we (only) want to answer a users' query, we do not need a full-out mapping for every node inside the refinement graph.

6.1.3. Matcher Mode

Once both refinement graphs $\mathbb{G}_{\mathrm{ref},s}$ and $\mathbb{G}_{\mathrm{ref},t}$ and have been generated, we need to derive correspondences between states. Because states represent conjunctive terms, the result of the mapping is then a set of tuple generating dependencies, which represent our desired, refined, complex mapping. The correspondence between pairs of completion graphs and tgds has been developed in chapter 5.

In general, there are $O(nm)$ possible mappings between n source and m target states, forming a complete bipartite graph. Because no information is gained when all mappings are produced, our goal is now to retain only those edges/mappings from the complete bipartite graph that are *relevant* for the alignment. An initial selection is provided by the *refinement strategy* or *matcher mode*.

A straightforward approach seems to be to consider the direction of the intended mapping: source elements are mapped to target elements. In this case, we need to find the most suitable (according to sim_G) mapping for each of the source nodes.

In general, it is possible to identify two different scenarios

known source In the first scenario, a fixed set of source refinements is to be matched against an unspecified set of target refinements. For example, when the target schema is relatively new and the user is more familiar with the previous data schema. It is then reasonable to expect the user to manually select parts of the source ontology that he wants to be transferred into the target ontology.

For example, in figure 6.1, selecting only the leaf nodes of the source tree simplifies matching, because a single source graph is matched against the target graphs and selection takes place only on the target side.

With *known source* matchings, the matching algorithm should transfer as much information as possible from the source to the target graph, but needs to take care not to "invent information" in the target graph. If a semantic feature of a source graphs cannot be properly mapped into a similar feature in a target graph, the feature should rather be ignored.

In most scenarios, *known source* should be the default matching scenario, as it tries to transfer as much information as possible from the source to the target ontology. Known source shall also the chosen matcher mode when ref_s is specified, because in this case, we match against known source terms.

known target In the preservation scenario, one can usually expect the reverse situation. The user extracting data from the archive is familiar with the target schema and can specify the parts of the target ontology that he wants to be extracted. In this case, the matcher must determine the parts of the source refinement graph that contains as much information as possible to "fill" the target graph.

In this case, the matching algorithm must select source graphs that contain as many of the semantic features of a single target graph. Once again, if a semantic feature of a source graph cannot be properly mapped into a similar feature in a target graph, the feature should rather be ignored.

Known target shall be the chosen matcher mode, when ref_t is specified. When neither ref_t nor ref_s are present, known target should be avoided as it will potentially generate many duplicate mappings for the same source element.

6.2 Refinement Rules

The final pieces of the refinement process are the definition of refinement rules P and the discovery of a suitable similarity function sim_G. Implementing sim_G will be discussed on its own in chapter 7. In this section, we will describe possible refinement rules.

When using a naive approach, defining refinement rules is rather simple: Any operation that adds to one of the sets, either V, E, or $\mathcal{L}$ of a completion graph either maintains the current interpretation or causes a narrowing of the interpretation of a completion graph. This is formalized in theorem 6.1.

Theorem 6.1 Completion Graph Refinement

Given two consistent completion graphs $G_r = (V_r, E_r, \mathcal{L}_r)$ and $G = (V, E, \mathcal{L})$ with $V_r \subseteq V$, $E_r \subseteq E$, $\mathcal{L}_r \subseteq \mathcal{L}$, then $G_r \leq_{\text{ext}} G$ (see definition 4.11).

Theorem 6.1 is rather simplistic, it however shows that it is –again– simpler to operate on completion graphs than on abstract mapping rules. Starting from the most general graph $G_0 = (\emptyset, \emptyset, \emptyset)$ it is possible to iteratively generate any completion graph by using only three simple *refinement rules*:

add node Create a fresh node $z \in N$ and set $V \leftarrow V \cup \{z\}$.

add link Pick a role $r \in \mathcal{R}$ and two $x, y \in V$, and set $E \leftarrow E \cup \{r(x, y)\}$.

add concept Pick a node x and a concept $C \in \Sigma_L$ and set $\mathcal{L} \leftarrow \mathcal{L} \cup \{(x : C)\}$.

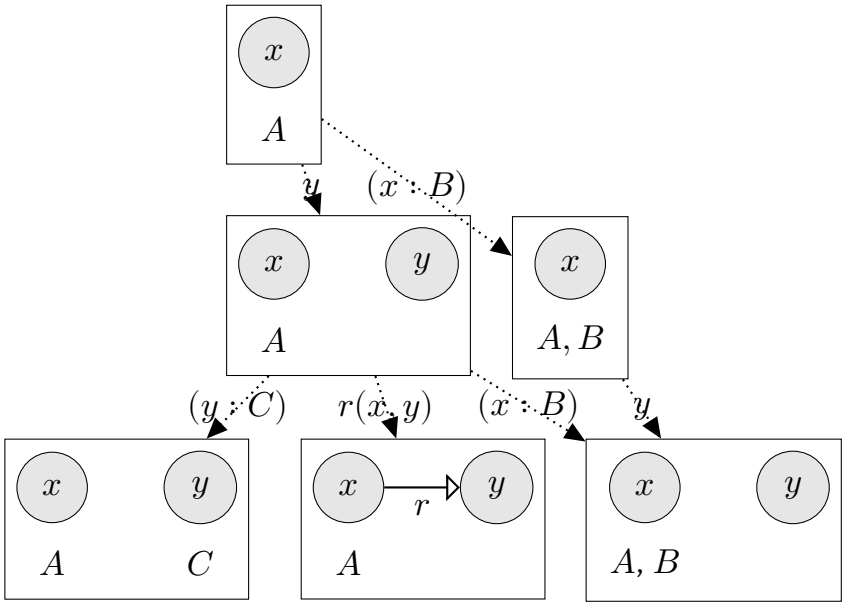
The result of each *refinement operation* is another completion graph with a "narrower" interpretation than all its ancestors, i.e. the refined graph G' is an extension of G: $G \leq_{\text{ext}} G'$ (see definition 4.11).

Figure 6.2 shows an exemplary refinement graph starting from an initial graph

$$G_0 = (\{x\}, \emptyset, \{(x : A)\})$$

Each completion graph is connected to at least one ancestor graph by the refinement rule that generated the graph. This creates a *graph of (completion) graphs*, the *refinement graph*. In the refinement graph, nodes are represented by (refined) completion graphs and the directed edges are the refinements that have been applied to generate a graph from its respective ancestor.

Figure 6.2. Refinement Graph



The DAG simplifies into a tree if a suitable partial order of application is imposed on the refinement rules. In the depicted graph, one of the paths $((x : B), y), (y, (x : B))$ in the right side of the figure is superfluous.

A natural extension to the primitive approach is to follow every application of a refinement rule by a run of the mapping tableau. In this case, refinements that lead to inconsistencies can be eliminated automatically. Because the tableau may generate multiple graphs from each refined graph (via $\sqcup$-branching), each graph node may now have multiple successors that are tagged with the same refinement rule. Additionally, because the tableau will usually insert additional, inferred concepts, the results of refinement rule applications is no longer orthogonal.

It is –unfortunately– not practical to implement this simple approach directly. A number of problems prevent the naive approach and mandate further refinement of the basic algorithm:

Search Strategy The three simple refinement rules are impractical: with most ontologies, the size of the generated refinement graph is not limited. When the ontology (and almost all ontologies do) allows for an infinite number of

individuals, the refinement graph also grows indefinitely. For practical use, the size of the refinement graph must be limited to manageable size. This problem will be addressed in the next section (section 6.2.1).

Semantic Refinement The second problem concerns the "semantic content" of the completion graphs generated by refinement. When starting with an initial completion graph, a goal of refinement is to introduce concepts and links that are actual "semantic refinements" of the initial mapping. This means, that the refined completion graph should not only be a logical specialization (in the form of a sub-interpretation) of the initial graph, but that the content of the refined graphs are also semantically subordinate to the initial graph. Because the refinement rules as presented so far operate almost completely agnostic of the existing completion graph, this property can easily be lost.

The situation can significantly be improved by modifying the refinement rules to be guided by the existing axioms in the completion graph and its associated TBox. Refinement rules that implement this feature will be developed in the next section (section 6.2.1).

6.2.1. Semantic Refinement Rules

The observations in the last section lead to the conclusion that a set of "smarter" refinement rules, that are based on an evaluation of the existing model graph and its TBox is needed. These refinement rules should also be "limited" in the sense that they do not generate infinitely large completion graphs that have little in common with the starting point of the refinement.

Scharffe [Sch09] defines an extensive set of *correspondence patterns*. The thesis lists multiple patterns for class, property, and attribute (i.e. data property) correspondences. Scharffe's list is detailed and often describes very high level transformations. For example, the "Class Correspondence by Path attribute Value" [Sch09, p. 184, sic] pattern can be represented –in the refinement framework– as a sequence of new node/add link operations followed by the introduction of the attribute value reference in the last node on the generated path. Some other

correspondence patterns are useful, but very hard to detect automatically. For example, the "Property Partition by Value Pattern" [Sch09, p. 228] splits a property into multiple properties based on an arbitrary condition on its values.

For example, in built heritage, an ordinal classification of damage severities (e.g. very good, good, ok, bad, very bad) is used in some ontologies, while other ontologies give the damage assessment as a quantified measurement (e.g. damage area in mm). While there is work to derive even such mappings between enumerated and continuous scalar ranges (e.g. [Doa02]), such refinements are often domain specific and it is only possible to obtain them automatically in rare cases. Nonetheless, complex transformations can still be added as additional refinement rules. This makes it possible to adapt the refinement process to domain specific requirements.

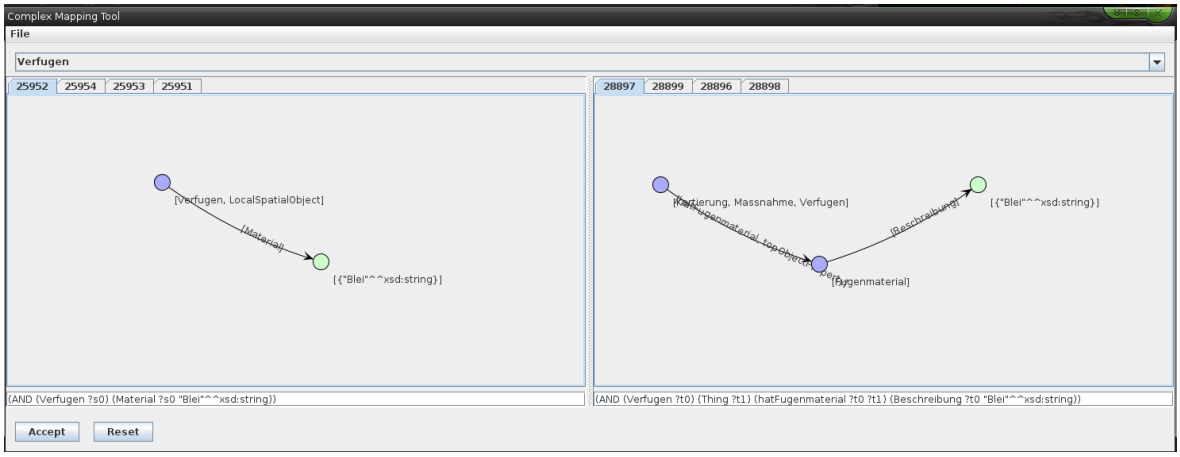
It is, however, still possible to cover a significant portion of Scharffe's correspondence patterns using two simple refinement rules. Both rules have preconditions that ensure, they are only applied when there is sufficient *evidence* already present in the existing completion graph.

The two refinement rules have been tested empirically. The manual mappings used for evaluation (section 8) have been generated using a purpose-designed mapping tool. The mapping tool (figure 6.3) allows to apply refinement rules for each node manually. Mapping generation for all concepts in the evaluation ontology (Bamberg Cathedral) were generated with only the use of the manual mapping tool that was developed as part of this thesis to facilitate the evaluation of the refinement approach.

6.2.2. Subclass Refinement

The first rule captures the scenario, when the directed refinement is a subclass of some (or more) existing concepts in the completion graph. This –for example– captures Scharffe's "Class by Attribute Value" and similar correspondence patterns (with appropriate refinement at the opposite side of a mapping). A concrete example of the pattern has already been discussed: the stone type subclasses in figure 1.4 can be generated using the subclass refinement rule.

Figure 6.3. Manual Mapping Tool: Mapping for Grouting



Mappings are generated interactively by selecting from a list of applicable refinement rules for each node. The mapping is applied automatically and the resulting refined graph(s) is/are displayed.

The rule generator is presented in function subClassRefinement, (function 6.1). For the subclass refinement rule, the TBox is scanned for possible candidates that could be inserted into the axiom set of an existing node. Candidates are only considered, if there is evidence that the subclassing is applicable here. The subsumption test in line 3 is performed using LillyTab (chapter 4). Effectively, the test checks if the second part of a GCI (i.e. D in $C \sqsubseteq D$, see section 4.2.2) is entailed by the axiom set $\mathcal{L}[x]$ of a node x. If this is the case, it is treated as evidence that C is a refinement candidate for the current node.

6.2.3. Role Successor Refinement

The second refinement rule combines the "add node" and "add link" operations from the initial, primitive ruleset. Combining "add node" and "add link" makes sure that the resulting completion graph is *connected*. Once again, the preconditions ensure that the rule only fires if there is evidence that the inspected node x might have a role successor. In this case, a role successor is tentatively added to a node, if the axiom set of the node is in the role's domain (line 4 in function roleSuccRefinement (function 6.2)). A role successor is also added, if the axiom set of a node contains a $\forall$-restriction, but no applicable successor is present

Function 6.1: subClassRefinement(G, TBox, x)

Input: $G = (V, E, \mathcal{L})$ a DL completion graph
TBox the TBox for G
$x \in V$, a node in G
Output: R: a set of refinement rules
begin

```
1   R ← ∅;
    /* Scan TBox for candidates                                    */
2   foreach (C ⊑ D) ∈ TBox do
        /* Check for evidence for subclass introduction            */
3       if G ⊨ (x : D) then
            /* Add candidate axiom                                 */
4           R ← R ∪ {(x : C)};
        end
    end
5   return R;
end
```

(line 6). Role successor refinements are executed simply by insertion of a suitable $\exists$-restriction. After inserting the new axiom, the next tableau completion step will eventually generate a suitable successor.

To prevent indefinite expansion, role successors are limited to a single successor. This is never a problem for functional roles, which make up the prevalent role type in the built heritage ontologies. If more than a single successor is necessary, a suitable limitation of the number of successors must be imposed by other means.

As an optimization, when ref_s (ref_t) is specified, only roles that appear in ref_s (ref_t) are considered as candidates.

6.2.4. Refinement Performance

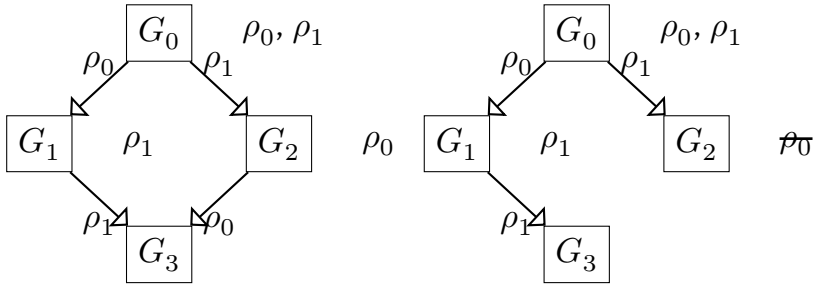
The primitive refinement process even with only subclass (section 6.2.2) and role successor (section 6.2.3) rules is too slow to be used in practice. Testing the preconditions for the refinements requires several reasoner calls per test. For example, an evaluation run on a moderately sized ontology (35 declared classes, 83 data properties) creates more than $135 * 10^6$ branches (i.e. completion graph copies)

Function 6.2: roleSuccRefinement(G, TBox, x)

Input: $G = (V, E, \mathcal{L})$ a DL completion graph
TBox the TBox for G
$x \in V$ a node in G
Output: R: a set of refinement rules
begin

1 $R \leftarrow \emptyset$;
`/* Scan through candidate roles */`
2 **foreach** $r \in \mathcal{R}$ **do**
3 $\mathsf{Domain}(r) \leftarrow$ the intersection of concepts explicitly asserted to be in the domain of r;
`/* check if the role's domain is implied by the node's axiom set */`
4 **if** $G \models (x : \mathsf{Domain}(r))$ *and* $\nexists y \,.\, r(x, y) \in E$ **then**
`/* add candidate axiom for role introduction */`
5 $R \leftarrow R \cup \{(x : \exists\, r \,.\, \top)\}$;
end
`/* check for universal restriction in the node's axiom set */`
6 **if** $G \models \forall\, r \,.\, C$ *and* $\nexists y \,.\, r(x, y) \in E \wedge G \models (y : C)$ **then**
`/* add candidate axiom for role introduction */`
7 $R \leftarrow R \cup \{(x : \exists\, r \,.\, C)\}$;
end
end
8 **return** R;
end

Figure 6.4. Ordering of Refinement Rules



(a) no ordering **(b)** ordered

Rule ordering indicates $\rho_0 < \rho_1$. $\not{\rho_0}$ means ρ_0 is marked not applicable for the respective completion graph.

and takes several hours to complete. This lacklustre performance is clearly unacceptable for an interactive tool and significant performance enhancements are necessary to make the refinement approach viable.

Rule Ordering

One important step that does not change the semantics of the refinement process but (drastically) increases performance is *rule ordering*.

In fig. 6.4a no order is applied to refinement rules. Because the refinement process is merge monotonic (definition 6.2, the order of rule application does not matter. Hence $[\rho_0, \rho_1]$ and $[\rho_1, \rho_0]$ both produce the same result graph. Without ordering, both paths are traversed.

When rule ordering is applied (fig. 6.4b), a rule with a higher order cannot be applied after a rule with a lower order. In this case ρ_0 has a higher order than ρ_1, hence after applying ρ_1, ρ_0 gets marked inapplicable (represented by $\not{\rho_0}$).

Rule ordering may be imposed arbitrarily as long as the ordering forms a total order.

Applicability of Rules

As noted in section 6.2.1 each refinement rule has a set of preconditions. For example, a subclass rule ρ might require that $C \sqsubseteq D \in \mathsf{TBox}$ as well as $D \in \mathcal{L}[x]$ for some node $x \in V$. If both preconditions are satisfied, we say that we have *evidence* for introducing C at x, i.e. evidence for applying ρ.

Evidence, however, is different from the *applicability* of a rule. For example, if both $\{D, \neg C\} \subseteq \mathcal{L}[x]$ we have evidence for introducing C, but we also immediately trigger a clash if we do so. In this case, we say that ρ is *not applicable*. If applying ρ does not lead to a clash, ρ is *applicable*.

These observations give rise to a tri-state logic for refinement rules. For some completion graph $G = (V, E, \mathcal{L})$, a graph node $x \in V$, and a refinement rule ρ, ρ can be in one of three states. For some node x and a refinement rule ρ, ρ either

has no evidence if the preconditions of ρ are not satisfied at x.

is applicable if the preconditions of ρ are satisfied and applying ρ does not create any inconsistencies.

is not applicable if either the preconditions of ρ are not satisfied or applying ρ creates an inconsistent completion graph $G[\rho]$.

Checking the state of a refinement rule is relatively expensive, because a consistency check (i.e. a reasoner call) is required. The check also seems necessary since the state of a rule can change during refinement. When D is not initially present in $\mathcal{L}_0[x]$ but subsequently becomes inserted, the state of ρ changes from *no evidence* to either *applicable* or *not applicable*.

Definition 6.2 Merge Monotonic

Given two completion graphs $G = (V, E, \mathcal{L})$ and $G' = (V', E', \mathcal{L}')$, with axiom language L, a transformation $\rho : G \mapsto G'$ is **merge monotonic** iff there is a node merge function σ such that

$$\forall\, x \in V, \phi \in \Sigma_L \,.\, G \models (x : \phi) \Rightarrow G' \models (x[\sigma] : \phi)$$

We also say that $(x : \phi)$ holds in both G and G' **modulo merging**.

One characteristic of the refinement process with the proposed refinement rules is, that it is *merge monotonic* (definition 6.2). Colloquially, this means that any concept that could be derived at some node in a graph G can also be derived in all its successor graphs G' modulo node merging. Because of this property, not all state transitions are possible for refinement rules. In particular,

1. a subclass refinement that has no evidence may become applicable or not applicable.

2. a subclass refinement that is applicable cannot loose its evidence. This directly follows from the merge-monotonicity of the termset of the candidate node x. D cannot vanish from x's termset.

3. a subclass refinement that is not applicable remains not applicable.

   The subclass refinement ρ is based on a GCI $C \sqsubseteq D$. If ρ is not applicable, it means that inserting C in $\mathcal{L}[x]$ results in a clash.

   The clash may either be local at x or it may happen at some other node. Non-local clashes can happen because of either an $\exists$- or a $\forall$-subterm in C.

   - A local clash happens if $\neg C \in \mathcal{L}[x]$. Because the refinement process is merge-monotonic neither x or any merge-successor of x can have $\neg C \notin \mathcal{L}[x]$. Local clashes are thus persistent.

   - If the non-local clash was because of a $\forall$-term, the clash must have happened, because some term has been introduced into the node set of an r-successor y of x (for some role r). Even after merging, x has only r-successors with monotonically growing axiom sets.

     Hence, we get either a local clash at y or a non-local clash at y. By induction over the term length, $\forall$-clashes are persistent during refinement.

   - If the non-local clash was because of an $\exists$-term, either the $\exists$-term is itself inconsistent or it introduces a clash in an r-successor of x. If the $\exists$-term is consistent by itself, this r-successor of y must be unique after merging, as otherwise we could simply introduce a fresh, consistent r-successor.

Since all potential r-successors y of x have monotonically growing axiom sets, local clashes at y are persistent. Non-local clashes at y can be shown to be persistent by induction on the nesting depth of the $\exists$-term.

4. a role successor refinement that is not applicable remains not applicable.

   For the same reasons as for subclass refinements, the preconditions for a role successor refinement cannot vanish without causing a clash. The refinement is non-applicable if the target node x already has an r-successor with either an existing set of axioms (which cannot vanish because of merge-monotonicity) or because introducing the $\exists$-term would cause a clash (which also cannot be undone because of merge-monotonicity).

5. a role successor refinement that has evidence can loose its evidence only because of a new r-successor.

   A role successor refinement has evidence, if either

   - $\mathsf{Domain}(r) \sqsubseteq \mathcal{L}[x]$ or
   - $\forall\, r\,.\,C \in \mathcal{L}[x]$.

   Since these are subsumption tests, the same criteria as for the subclass evidence can be applied. This part of the evidence is thus persistent.

   The role successor refinement also requires that x does not yet have suitable r-successor. Since r successors remain and their axiom sets grow monotonically, it is sufficient to check only for clashes at r-successors.

These observations can be used to significantly improve the performance of the refinement process. Where a state transition is not possible, the preconditions of a refinement rules often need not be re-checked. Where the preconditions of a refinement rule are partially invariant, only the variable part of the precondition must be re-checked. In particular,

- refinement rules with *no evidence* need to be re-tested,
- if a refinement rule is not applicable, it remains so. It can thus be ignored for the remainder of the refinement process,

- if a subclass introduction refinement based on $C \sqsubseteq D$ is applicable, subsequent steps only need to test consistency of $\mathcal{L}[x] \cup \{C\}$. Inconsistency with D is only possible if the completion graph is already inconsistent before the refinement, and

- if a role successor refinement is applicable, the next step only needs to test for consistency of $\mathcal{L}[x] \cup \exists r...$ as any precondition consistency checks have already been performed and cannot be invalidated.

The result of introducing these additional checks is an improvement of refinement performance by a factor of 5.33. In practical numbers, a full refinement had its running time reduced from slightly over 80 to only 15 minutes.

6.3 Interactive Alignment

With the results obtained so far, we are able to perform *interactive, assisted refinement*. As a proof of concept, a manual mapping tool has been developed to showcase the feasibility of the assisted refinement process.

The interactive mapping tool takes as input two ontologies and a simple reference alignment. It then allows the user to interactively refine each initial correspondence. The basic workflow can be outlined as follows:

1. The user first selects one of the initial correspondences from a drop-down menu. For example, in fig. 6.5, the correspondence for Vierung from the introductory example (section 1.2) has been selected.

   The possible correspondences are presented in the usual (also used by e.g. COMA++ [ADMR05], Snoggle [RDB+08]) side-by-side view with the source graphs at the left and the target graphs at the right.

   Both sides are presented as (tabbed) lists of (automatically layouted) completion graphs.

2. After selecting the initial alignment, the tool automatically starts a search for possible refinements. Once found, the possible refinements are made available to the user via context menu on the completion graph nodes as shown in fig. 6.6. Refinements can be applied to both the source and the target side of a displayed correspondence.

   On both sides of the tentative alignment, multiple completion graphs may be displayed. These will be shown separately on different tabs and can be selected by the user. Refinement of the different completion graphs is independent of the other graphs in the current suggestion list.

3. Selecting a particular refinement applies this refinement to the active completion graph and restarts the refinement search at the freshly obtained, refined graphs.

   Manual refinement is then repeated until the user is satisfied with the resulting correspondence.

4. Once the user is satisfied with a correspondence (represented by two side-by-side completion graphs), she can commit the rule and it will be saved as a new reference alignment.

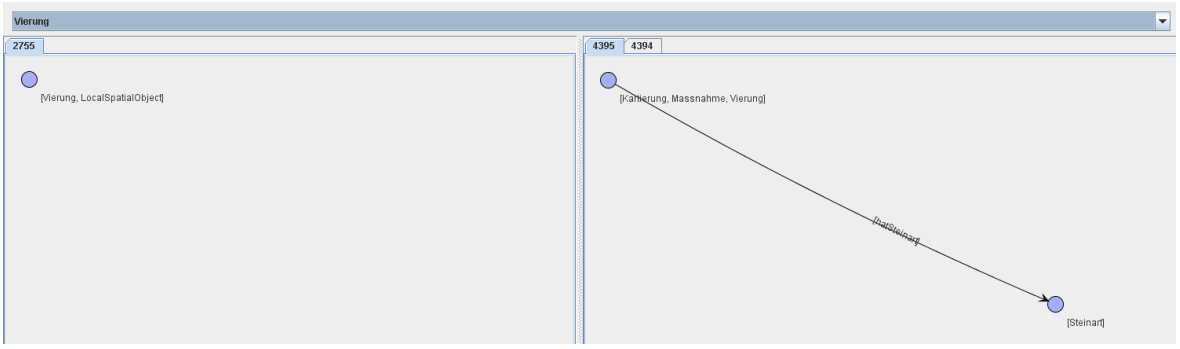
The whole process is repeated until the user is unable to determine more correspondences or until there are no remaining suggestions. For example, fig. 6.7 shows a possible correspondence for Vierung from the introductory example with refinement material type Coburger.

Interactive generation of complex alignments will be evaluated in section 8.2. In particular, all reference alignments used within this thesis have been generated using the manual refinement tool.

Chapter Summary

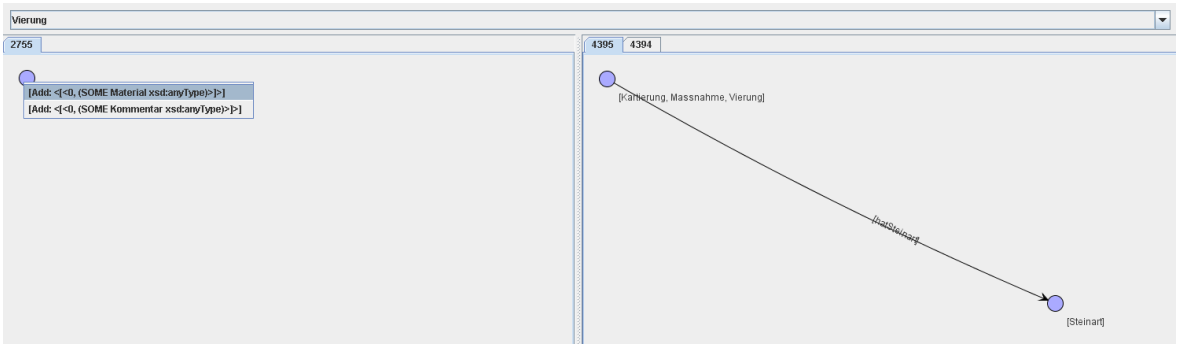
In this chapter, we have developed a method to obtain complex correspondences from simpler ones by defining an abstract, iterative *refinement process* (section 6.1). The refinement process takes as input a set of alignments and produces an interlinked set of possible refinements of the original mappings, the refinement graph.

Figure 6.5. Interactive Mapping Tool, generating mapping for Vierung



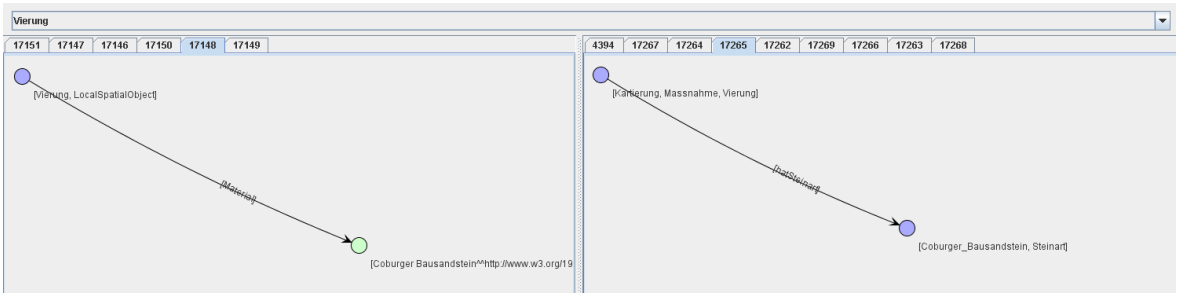
Source view is on the left, target view is on the right. Graph parts automatically expanded by the reasoner are shown. Multiple graphs are displayed in different tabs.

Figure 6.6. Interactive Mapping Tool, Refinement suggestions for Vierung



Source view is on the left, target view is on the right. Refinements can be applied by selecting from a context menu. The context menu is specific to each graph node, allowing local control over the refinement by the user.

Figure 6.7. Interactive Mapping Tool, Possible complex correspondence for Vierung with material Coburger



Source view is on the left, target view is on the right. The user has selected a plausible correspondence and can now commit that correspondence as a new mapping rule.

The process is able to incorporate user input in the form of reference queries ref_s and ref_t. We have explored initial strategies (section 6.1.3) to direct the refinement process based on these reference queries.

In a next step, we have instantiated the abstract process into a concrete version using DL completion graphs as the formalism for refinement state representation. In this, we incorporated results from the previous chapter (chapter 5) that enabled us to represent complex mappings using (augmented) DL completion graphs. This model-based implementation of the refinement process also enabled us to develop two relatively simple *semantic* refinement rules that make use of information contained in TBox of both the source and target ontologies.

To show the feasibility of the completion graph based refinement approach, we showcased a tool to perform interactive refinement based on the presented refinement rules.

In the next chapter, we will build on these results and develop techniques to extract suggestions for (refined) complex mappings from model-based refinement graphs. In particular, we explore different implementations of the state similarity function sim_G (previously sim_S).

SECTION 7.0

Comparing DL Completion Graphs

In the previous chapter, I have developed a method to derive more complex "refined" alignments from initial simple alignments.

Having a completion graph representation of an alignment, however, is not beneficial on its own. In this chapter, the second part of the core of this thesis is presented: methods to compare and align DL completion graphs to obtain a complex ontology alignment.

Different similarity measures for DL completion graphs will be iteratively developed to obtain improved alignment results.

Model based matching makes it possible to make use of information explicitly present in the completion graphs to define relatively simple semantic refinement rules (section 6.2.1) that make use of local information but also cover a variety of possible refinements.

The main benefit, however, is the ability to make use of the explicit information contained in the completion graph and determined during the refinement process to improve selection of mapping rules. We have already established (section 5.2) that a modified tableau algorithm is useful to relate tuple generating dependencies (tgds) and pairs of DL completion graphs. This modified tableau is one of the reasons for the implementation of the LillyTab reasoner (section 4.3). Because of this result, it seems reasonable to extend use of semantic information also to the comparison of completion graphs.

Table 7.1. Simple Matching Results for the "Dom Bamberg" Ontology and its Evolved Successor

matcher	#	prec. %	rec. %	f %
1. vector-based multi-words	49	55.1	26.7	36.0
2. lexical synonyms	39	59.7	22.8	32.9
3. parametric string	53	54.7	28.7	37.7
4. similarity flooding	19	9.1	1.0	1.8
5. AnchorFlood	47	42.6	19.8	27.0
6. AROMA	40	67.6	24.8	36.2
7. Blooms	12	41.7	5.0	8.8

Previous research in this direction indicates, that comparing saturated graphs is indeed beneficial with regard to similarity calculation. For example, Janowicz and Wilkes [JW09] proposed SIM-DL$_A$, which uses tableau expansion of complex $\mathcal{ALC}$ geospatial concepts to determine the similarity between the concepts. Their algorithm is largely based on traversal of the completion tree ($\mathcal{ALC}$ has the tree model property) and comparing individual branches independently. Because of their promising results, it seems reasonable to assume that comparing DL completion graphs is also possible using relatively simple methods for more complex logics.

7.1 Requirements Analysis

A "mapping capable" similarity measure between DL completion graphs needs to consider several desirable properties that can be derived from the structure of the ontologies involved and from the refinement process.

7.1.1. Element Level Similarity

All matching systems are based at some point on an element-level similarity measurement, which yields the degree of correspondence between concepts, roles and individuals. Even purely structural systems like Similarity Flooding [MGMR02] or AnchorFlood [SA08] need an element level matcher to bootstrap.

As described in section 6.1.1, model based refinement also requires simple $1:1$ *anchoring* correspondences to start the refinement process. To obtain the initial alignment for model based refinement, AgreementMaker [CAS09] was used as a mapping and evaluation tool. The results are shown in table 3.1 (duplicated in table 7.1 for easier reference). Because complex matching is required to properly align both ontologies involved, the matching quality is below what is typical for these matchers (i.e. $\geq 90\%$ precision).

The table, however, shows another interesting fact: systems that are purely based on the lexical similarity (vector-based multi-words, lexical synonyms, parametric-string, AROMA [DE08]) perform better than their counterparts that also use structural (AnchorFlood, similarity flooding [MGMR02]) or external (Blooms) [JHS+10] methods.

A similar property has been observed by Ghazvinian et al. [GNM09] in a different domain: matching biomedical ontologies does not necessarily benefit from complex matching algorithms but can often be performed using relatively simple methods. The same seems to be true for built heritage. Built heritage jargon is highly specialized and specific meanings are associated with certain technical terms. Typical methods like web search [PSCC10] or WordNET similarity [VVR+05] do not work very well for such highly specialized jargon. On the other hand, a complex lexical similarity function is often simply not necessary: jargon changes only slightly over time, at least for a specific site. Lexical semantics in built heritage are thus very homogeneous and persistent.

Because of this observation, lexical similarity will also be used as the foundation for all similarity measurements between completion graphs (sim_G). At this point, it seems sensible to try to re-use the element-level similarity measurement directly for sim_G. Unfortunately, this is usually not possible off the shelf, because existing systems are not prepared to take DL completion graphs as input and return a global similarity measure between both graphs. Consequently, we need to provide our own implementations of sim_G.

Table 7.2. Alignment Length

	$G_{s,1}$	$G_{s,2}$	$G_{t,1}$	$G_{t,2}$
A	✓		✓	✓
B	✓	✓	✓	

Because we want to use lexical comparison, we first need a method to extract relevant phrases from completion graphs. The core information content of a augmented completion graph $G = (V, E, \mathcal{L}, \kappa, \mathsf{Dep}, \mathsf{Mer})$ is contained in the sets $\mathcal{L}$ and E, enhanced by additional information from κ and Dep. Since $\mathcal{L}$ contains axioms $\mathcal{L} \subseteq \Sigma_L$ (with $L = L_{\mathcal{SHOIF}(\mathcal{D})}$), we base our similarity measure on the similarity of axiom sets.

Hence, our completion graph similarity measure $\mathsf{sim}_G : G^2 \mapsto \mathbb{R}_0^+$ will be based on a axiom based similarity measure $\mathsf{sim}_L : {\Sigma_L}^2 \mapsto \mathbb{R}_0^+$.

7.1.2. Maximum Information Transfer and Alignment Length

The *overall length* of the alignment is a factor with regard to the quality of an alignment.

Have a look at table 7.2 and assume that A and B are semantic features of the respective completion graphs $G_{s,i}$, ($G_{t,j}$, respectively) . It should be easy to see, that $G_{s,1}$ matches best with $G_{t,1}$ and $G_{s,2}$ matches best with $G_{t,2}$.

There is also an ordering between the alignment pairs: $e_{2,2} = \left(G_{s,2}, G_{t,2}\right)$ is a refinement of $e_{1,1} = \left(G_{s,1}, G_{t,1}\right)$. Any knowledge base fragment that is mapped by $e_{2,2}$ automatically also contains a subfragment that is an application of $e_{1,1}$.

This also means that an alignment like $e_{1,2} = \left(G_{s,1}, G_{t,2}\right)$ is only *incomplete,* but not necessarily *incorrect.* Because the source side is too restrictive for $e_{1,2}$ it is possible that not all information that could be mapped will be mapped. However, all information at the target side is still justified by a source feature (correctness). Nonetheless, $e_{1,1}$ should be considered the better alignment, because it transfers more information from the source to the target than does $e_{2,2}$.

Following this argument, we can deduce, that

- the alignment should aim to produce the longest correct alignment, and
- it should be possible to either automatically produce all correct, shorter alignments or to automatically derive them from the longest correct alignment.

Consequently, when the similarity computation results in a tie between possible options, we prefer the longer refinement rule. The length of a refinement rule is measure in the number of conjunctive terms that it contains in both its body and head.

7.1.3. Avoiding Unjustified Elements

Striving for maximum alignment length, alone, however, is also insufficient. To demonstrate the problem, consider again the introductory example from section 1.2. At some point, the refinement graphs source side and target side graphs in figure 6.1 have been generated. The initial mapping specifies, that $G_{s,0}$ maps to $G_{t,0}$, which is our starting point. Similarly, $G_{s,1}$ (plaster) corresponds to $G_{t,3}$ (P), $G_{s,2}$ (brick) with $G_{t,4}$ (B) and $G_{s,3}$ (lead) with $G_{t,5}$ (L).

Matching $G_{s,4}$ (other), however, is a slight problem. The material type "other" is not present in the target ontology. Its presence in the source ontology is actually a modelling error. If the material type for a grout filling is not known, the respective property value should have been simply omitted instead of introducing an additional "catch-all" property value. $G_{s,4}$ can either be mapped to $G_{t,0}$ or $G_{t,1}$, respectively.

$$
\begin{aligned}
G_{s,0} &\mapsto G_{t,0} \text{ or } G_{t,1} \\
G_{s,1} &\mapsto G_{t,3} \\
G_{s,2} &\mapsto G_{t,4} \\
G_{s,3} &\mapsto G_{t,5} \\
G_{s,4} &\mapsto G_{t,1} \text{ or } G_{t,0}
\end{aligned}
$$

$G_{s,0} \mapsto G_{t,0}$ is the graph pair implied by the initial correspondence before refinement. In source-to-target mode, it is necessary to find a correspondence for every source graph.

Some similarity measures consider only positive correspondences and simply ignore features that are present in one but not on the other side of the comparison. For example, primitive cosine similarity (without length normalization) does not take into account vector space component axes with a zero value in one of the document vectors. For alignment generation, this is undesirable. The problem appears on both sides, of the mapping:

- Expressing a feature on the target side that has no justification on the source side is equivalent to inventing information without a basis in existing data.
- Having an *over-refined* source part makes the query side of a complex correspondence too restrictive. This means that the alignment might miss out on mappable information that is present in the source knowledge base because that part of the knowledge base is filtered out by the over-refined source.

Consequently, while –in general– larger alignments should be preferred over shorter alignments, this is only true if both sides of a correspondence contain only elements that are justified by elements on the other side of the correspondence.

7.1.4. Phrase Extraction

Because we want to make use of lexical similarity, we need a method to extract relevant textual phrases from completion graphs. Formally, we need a procedure that takes as input a completion graph G and returns a *multiset* of phrase tokens T. The tokens usually are, but need not be represented by (Unicode) character strings with maybe attached metadata. We use the term token or phrase interchangeably. As noted in section 7.1.1, our completion graph similarity measure sim_G will be based on the axioms in $\mathcal{L}$.

The calculation of the similarity measure will proceed in two steps:

1. *select* or *filter* relevant axioms from $\mathcal{L}$ and then

2. *extract* a multiset Ph of phrases from the filtered terms.

The phrase extraction step needs to be performed by every lexical matching system, however the exact procedure is rarely described in full detail in the literature. Even dedicated papers like [SMW15] tend to skim over the details of lexical extraction, leaving out algorithmic details.

This is unfortunate, as the exact procedure of phrase extraction from an ontology has a strong impact on alignment quality. Small changes can alter the quality of the result by two digit percentages. Leaving out the description of the extraction algorithm from a matcher system description greatly reduces the reproducibility of published results.

To describe an ontology phrase extractor, at least the following information is needed

granularity of extraction Is the extraction based on single elements (concepts, roles), on fragments of an ontology or on sets of axioms? Are only TBox elements used for extraction or are individuals considered during analysis?

resource specification Which resources are used for phrase extraction?

Encoding of semantic web ontologies is typically done using web standards, most often XML [PPSM09]. This means in particular, that element names are usually represented by Internationalized Resource Identifiers (IRI, [DS05]). During phrase extraction, some systems consider only the IRI [DS05] of an element, most often only the fragment (`#...`) or last path component (`/Ashlar`). This is conforming to most modelling conventions but rarely described explicitly in matcher descriptions. Indeed, e.g. AROMA made the implicit assumption that every IRI that did not have the ontology IRI as a prefix can be ignored completely. Lifting this restriction would confuse the matcher because the triple store's query engine returned IRIs from imported ontologies as local elements and thus confuse the matcher.

On a different note, ontology representations often contain a mix of absolute and relative IRIs. Missing normalization would result in the assumption that relative IRIs are shorter than they actually are, possibly skewing matcher results.

Does token extraction consider annotations (e.g. `rdfs:label`)? Are annotations used unconditionally or only if token extraction from the IRI did not yield satisfactory results?

preprocessing How are tokens preprocessed (if at all)?

Which tokeniser is used? Is the original token also returned or only the decomposed result? Is Unicode normalization or language specific normalization (e.g. German "ß" ⇒ "ss", "ä" ⇒ "ae") performed?

Are tokens subject to lexical and morphological analysis (e.g. stemming)?

Which other processing is applied (e.g. compound word decomposition)?

special processing Are any other specialized analysis methods employed?

This thesis uses a few hand-crafted together with as many off-the-self extractor components as possible.

1. The extractor takes as input an ontology axiom $ax \in \Sigma_L$

   a) In the simplest case the axiom is a class assertion ($C(x)$) and the class reference C will be used for further processing.

   b) if the axiom is a role assertion ($(\forall\, r\,.\, C)(x)$, $(\exists\, r\,.\, C)(x)$), tokens are extracted from both the role element r as well as recursively from the subterm C.

   c) if the axiom is a literal reference, that is either a nominal or a data value, the referenced individual is processed further.

   d) other axioms are ignored.

2. In a next step,

a) for any named entity (concepts, roles, individuals), the IRI is inspected.

   If a fragment part (#) is present and the string tokeniser (see below) does not yield an empty result for the fragment string, the fragment part is fed to the tokeniser. Otherwise the full IRI is processed.

   This is a heuristic that reduces number of phrases with low information content early on, because most ontology element IRIs will have a common prefix (usually that of the ontology IRI).

b) Additionally, if the input entity has any annotations with a literal value (e.g. `rdfs:label`), the literal's value is fed to the string tokeniser.

c) Any data value is preprocessed. Some ontology access APIs attach a type specifier (e.g. `^^xsd::string`)) to literal values. This type specifier is removed together with any surrounding quotation marks.

3. The resulting string tokens are fed to a special purpose tokeniser.

   a) The tokeniser first applies a thesaurus lookup. This thesaurus is user supplied and most likely needs to be adopted for a target domain. It also provides an easy entry point for user supplied information.

      Any synonyms returned from the thesaurus are fed to the tokeniser *in addition* to the original strings.

   b) The tokeniser always returns the original input string unmodified as a single token.

   c) Subsequently, the tokeniser splits the input phrase into sequences containing only Unicode letters (Unicode general category *Lu* and *Ll* as well as digits). Each token must start with at least single letter.

      This custom implementation is required, because the standard tokeniser from Apache Lucene does not handle decomposition of camelCased words, which are commonly used to distinguish individual parts in compound words and multi-word element names.

4. Each token is again looked up in the thesaurus, any resulting synonyms are added to the token stream in addition to the original phrase.

5. The output from the tokeniser is fed to a (linguistic) analyser. The analyser

   a) performs compound word decomposition using the standard Apache Lucene dictionary compound word filter and the OpenOffice dictionary as a word list,

   b) lower cases all characters,

   c) applies a stop word filter using a standard word set,

   d) applies lexical normalization using Lucene, and

   e) performs stemming using the Lucene-supplied snowball stemmer[12].

   Apart from the ability to selectively disable individual steps, this configuration resembles the `GermanAnalyzer` filter from standard Lucene with the added ability to selectively disable different parts of the analyser and with a custom tokeniser.

To evaluate the effects of the lexical processing steps, the empirical evaluation will be run in three different modes:

stemmer Runs only the stemmer without compound word decomposition or normalization.

normalizer Runs the stemmer and the normalizer without compound word decomposition.

stemmer-decomposer Runs all three steps: decomposer, normalizer, stemmer.

The effects of different analyser configurations on alignment quality are shown in section 8.3.2.

[12] `org.apache.lucene.analysis.de.GermanLightStemFilter`

Definition 7.1 Extracted Phrase Set

Given a completion graph G, a (possibly infinite) set of distinct phrases $\mathbb{Ph}$, an axiom filter function filter : $\mathbb{G} \mapsto {\Sigma_L}^+$, together with a phrase extraction function extractPhrase : ${\Sigma_L}^+ \mapsto \mathfrak{P}(\mathbb{Ph})$, the **extracted phrase set** or **extracted token set** $\mathsf{Tk}(G)$ is the multiset

$$\mathsf{Tk}(G) \equiv_{\text{def}} \bigcup_{ax \in \mathsf{filter}(G)} \mathsf{extractPhrase}(ax)$$

$\mathsf{Tk}(G)$ may also be written only as Tk, when G is clear from the context.
When multiple completion graphs G_i, ... are involved, we also use Tk_i, ... as shortcuts for simplicity.

Function 7.1: extractPhrase(x, $\mathbb{O}$)

Input: $x \in (\mathcal{R} \cup \mathcal{C} \cup \mathcal{I})$ an element reference from one of the ontologies in $\mathbb{O}$
$\mathbb{O}$ a set of ontologies
Output: Ph: a set of extracted phrases

```
begin
1   | Ph ← ∅;
2   | if hasIRI(x) then
3   |   | iri ← getIRI(x);
4   |   | if hasFragment(iri) then
5   |   |   | Ph ← Ph ∪ tokenise(getFragment(iri));
    |   | end
    |   | else
6   |   |   | Ph ← Ph ∪ tokenise(iri);
    |   | end
7   |   | foreach an ∈ annotations(x) do
8   |   |   | Ph ← Ph ∪ tokenise(an);
    |   | end
    | end
9   | return Ph;
end
```

Function 7.2: extractToken(ax, $\mathbb{O}$)

Input: $ax \in \Sigma_L$ an axiom in $\mathcal{SHOF}(\mathbb{D})$
$\mathbb{O}$ a set of ontologies
Output: Ph: a set of extracted phrases
begin
1 Ph $\leftarrow \emptyset$;
/* ax is a class assertion */
if $\exists C \in \Sigma_L, \underline{a} \in \mathcal{I} \,.\, ax = C(\underline{a})$ **then**
2 Ph $\leftarrow$ extractPhrase$(C, \mathbb{O})$;
end
/* ax is a role axiom */
if $\exists C \in \Sigma_L, r \in \mathcal{R} \,.\, (ax = \exists\, r \,.\, C \vee ax = \forall\, r \,.\, C)$ **then**
3 Ph $\leftarrow$ extractPhrase$(r, \mathbb{O})$;
end
/* ax is an individual reference */
if $\exists\, \underline{a} \in \mathcal{I} \,.\, (ax = \{\underline{a}\})$ **then**
4 Ph $\leftarrow$ extractPhrase$(\underline{a}, \mathbb{O})$;
end
5 **return** Ph;
end

7.1.5. Axiom Filtering

Written out in full, the inputs for our to-be-designed similarity measures are two augmented completion graphs, a source graph $G_s \equiv_{\text{def}} \left(V_s, E_s, \mathcal{L}_s, \kappa_s, \mathsf{Dep}_s, \mathsf{Mer}_s\right)$ and a target graph $G_t \equiv_{\text{def}} \left(V_t, E_t, \mathcal{L}_t, \kappa_t, \mathsf{Dep}_t, \mathsf{Mer}_t\right)$ together with their respective ontologies O_s and O_t. Some similarity measures might also need access to the full set of completion graphs $\mathbb{G}_s$ and $\mathbb{G}_t$ that participate in the mapping.

As outlined in the previous section, we will use phrase extraction from DL axioms to facilitate the basic comparison of completion graphs. Hence all methods primarily focus on the two sets $\mathcal{L}_s$ and $\mathcal{L}_t$. On the other hand, not all axioms from $\mathcal{L}_s$ ($\mathcal{L}_t$) maybe considered relevant. We use *filtering* to extract only a limited subset of *relevant* axioms.

Two filtering approaches will be followed in this thesis:

no filtering Phrase extraction runs on unfiltered $\mathcal{L}_s$ and $\mathcal{L}_t$.

Function 7.3: analyseToken(tk, thes, decompose, lower, Stop, normalize, stem)

Input: $\mathsf{tk} \in \mathbb{Ph}$: an extracted token (from phrases $\mathbb{Ph}$).
thes: A function thes : $\mathbb{Ph} \mapsto \mathfrak{P}(\mathbb{Ph})$ returning zero or more synonyms for an input phrase ph. thes is assumed to be reflexive, i.e. $\mathsf{ph} \in \mathsf{thes}(\mathsf{ph})$.
decompose: A function decompose : $\mathbb{Ph} \mapsto \mathfrak{P}(\mathbb{Ph})$ returning the decomposition of some phrase ph
lower: A function decompose : $\mathbb{Ph} \mapsto \mathbb{Ph}$ returning the lower case of some phrase ph
Stop: A set of stop words.
normalize: A function normalize : $\mathbb{Ph} \mapsto \mathbb{Ph}$ returning the normalization of ph
stem: A function stem : $\mathbb{Ph} \mapsto \mathbb{Ph}$ returning the stemmed version of of ph

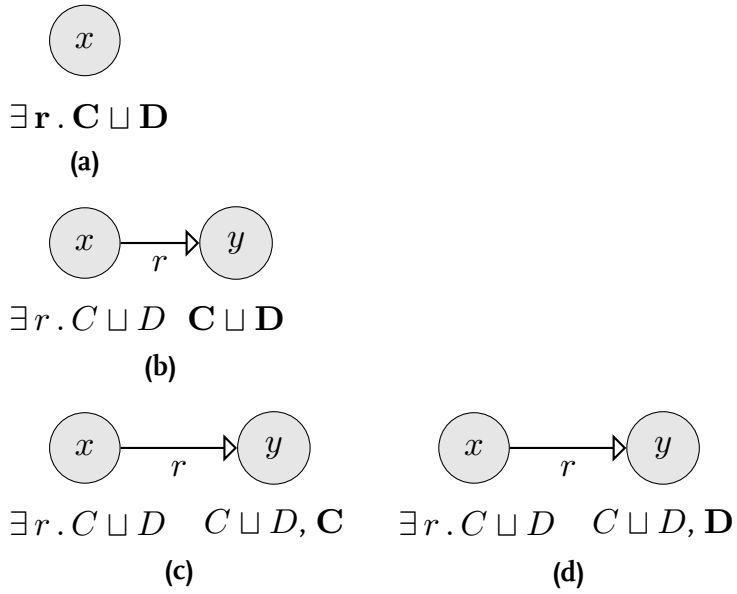
Output: Tk: a set of transformed tokens, maybe empty

begin

1 $\mathsf{Tk} \leftarrow \bigcup_{\mathsf{tk} \in \mathsf{Tk}} \mathsf{thes}(\mathsf{tk})$;
2 $\mathsf{Tk} \leftarrow \bigcup_{\mathsf{tk} \in \mathsf{Tk}} \mathsf{decompose}(\mathsf{tk})$;
3 $\mathsf{Tk} \leftarrow \{\mathsf{lower}(\mathsf{tk}) | \mathsf{tk} \in \mathsf{Tk}\}$;
4 $\mathsf{Tk} \leftarrow \mathsf{tk} \setminus \mathsf{Stop}$;
5 $\mathsf{Tk} \leftarrow \{\mathsf{normalize}(\mathsf{tk}) | \mathsf{tk} \in \mathsf{Tk}\}$;
6 $\mathsf{Tk} \leftarrow \{\mathsf{stem}(\mathsf{tk}) | \mathsf{tk} \in \mathsf{Tk}\}$;
7 **return** Tk;

end

Figure 7.1. Examples of Existential Generating Dependency Paths



κ-based filtering Phrase extraction is primarily based on the governing terms κ_s (κ_t). Additional axioms *related* to governing terms are also optionally considered.

7.1.6. EGD Path

One problem with governing terms is that in a saturated tableau, a governing term is never a role axiom and this causes role connections to be unrepresented in any similarity computation based purely on governing terms. One solution to this problem lies in back-tracing the origins of governing terms across the reasoning and refinement steps.

Consider, for example, the initial ABox in fig. 7.1a. The *initial axiom* is marked as a governing term in bold font. Figures 7.1b to 7.1c show the expansion steps performed by the DL tableau. In particular, it is possible to trace the movement of the governing term from $(x : \exists r . C \sqcup D)$ first to $(y : C \sqcup D)$ and then to both to $(y : C)$ and $(y : D)$.

Thus, if we look at the final outcome, $(y : C)$ depends on $(y : C \sqcup D)$, which in turn depends on $(x : \exists\, r\,.\, C \sqcup D)$. Equally, for $(y : D)$ to be present in the final completion graph, $(y : C \sqcup D)$ needs to exist and (transitively) $(x : \exists\, r\,.\, C \sqcup D)$ needs to be present as well. The sequence of axioms that need to be present for another axiom to exist is called the *dependency path* of the original axiom.

Tracing back the dependency path is useful, because it gives information about the origins of an axiom. This is especially important for mapping, because in a saturated tableau, all governing terms κ are atomic. A major drawback of this focus is that role connections are not represented at all. Evaluating the dependency path remedies this problem as all role axioms leading to a certain governing term are contained in the dependency path. The general idea of this path-tracing is not new. The Clio system [HMH01] implements a similar method, albeit only for (explicit) relations in relation databases. The dependency path is a similar concept. The axiom-based nature of the ontological model, however, makes the dependency path both harder to trace and more expressive.

However, given the existing mapping tableau, not all of the axioms on the dependency path contain additional mapping information. In figure 7.1b, $(y : C \sqcup D)$ is not useful when traced back from either $(y : C)$ or $(y : D)$, because we already have information about which one of C or D is present. A naive word extraction approach applied to $C \sqcup D$ could actually do harm with regard to completion graph comparison. If $(y : C)$ is already known, adding extracted words from D would dilute the already stricter distinction.

We can make similar observations for the other compound axiom types:

$(x : C \sqcap D)$ The axiom will be decomposed into two governing terms $(x : C)$ and $(x : D)$. No information is lost.

$(x : C \sqcup D)$ The axiom will be decomposed into two governing terms $(x : C)$ and $(x : D)$ *in two different completion graphs*. No information is lost. Considering $(x : C \sqcup D)$ might even be harmful.

$(x : \forall\, r\,.\, C)$ The axiom will either be ignored when no r-successor is present or it will generate a governing term $(z : C)$ for some existing r-successor z.

In both cases, we do not loose information about r. If no r-successor is present in the graph, the successor is not part of the information represented by the completion graph. If an r-successor is present, it was either present initially or generated by other means.

$(x : \neg C)$ As noted in section 5.1.3, negations are problematic only when the negated axiom is atomic (i.e. a class or nominal reference) as the negation will otherwise be expanded further.

No attempt is made to handle negations for atomic axioms and they are silently ignored, a precedence established by LogMap [Mei11].

$(x : \exists\, r\,.\, C)$ Both the mapping tableau as well as the refinement process only generate new completion graph nodes from existential axioms. Thus any node in the completion graph is connected to the initial root node via a sequence of at least one existential term dependency.

Because of the deterministic modification of the tableau (see section 5.1.6), generating terms are also unique for each target node unless the role is functional. However, when two generating terms for a functional role appear on the same node, e.g. $(x : \exists\, r\,.\, C)$ and $(x : \exists\, r\,.\, D)$ the source node x is still the same.

Given an augmented completion graph G and an axiom $ax = (x : C)$ that was introduced during the refinement process, determining the *dependency path* of ax is relatively simple, the dependency path is the transitive hull of all parents of ax in Dep.

Definition 7.2 Dependency Path

Given an augmented completion graph $G = (V, E, \mathcal{L}, \kappa, \mathsf{Dep}, \mathsf{Mer})$ and an axiom $ax \in \mathcal{L}$, the dependency path $DP(ax)$ is the transitive closure (least fixed point) such that

- $ax \in DP(ax)$
- $(ax \in DP(ax) \wedge (ax' \mapsto ax) \in \mathsf{Dep}) \Rightarrow (ax' \in DP(ax))$

The full dependency path, however, is a slightly unwieldy set as it contains many axioms that might either be harmful ($A \sqcup B$) when considered for mapping or yield no new information ($\forall\, r\,.\, A$, $A \sqcap B$). To remedy this problem, we can make a very strong observation: any node x inside a completion graph G as generated from the mapping tableau and/or refinement has

- either a governing term itself, i.e. $\exists\, C\,.\,(x : C) \in \kappa$, or
- has axioms that lie on the dependency path to some other governing term, because every leaf node in the saturated graph has at least one governing term.

Additionally, all graphs that appear during refinement share a common ancestor and they also share a common root node x_0. x_0 has the lowest possible position in the total order of generated nodes (or even is a named node), so every node merge operation (function augmerge 5.2) that involves x_0 happens in the direction of the root node. In other words, the root node is common to all graphs and is also persistent during refinement.

Because of this and because of how tableau completion and refinement operate, there is at least one sequence of dependencies of the form $egd = (x : \exists\, r\,.\, C \mapsto y : C)$ for every non-root node in G that terminates at the root node. For any given node y, any such path from y via such *existential generating dependencies* (definition 7.3) is called an *egd path*.

Definition 7.3 Existential Generating Dependency

Given an augmented completion graph $G = (V, E, \mathcal{L}, \kappa, \mathsf{Dep}, \mathsf{Mer})$ any entry $egd = (x : \exists\, r\,.\, C \mapsto y : C) \in \mathsf{Dep}$ is called an **existential generating dependency** or **egd**.
Given a root node $x_0 \in V$ and a node $y \in V$, a sequence of egd that, lead from y back to x_0 is called an *egd path*.

The definition of the egd path is more relaxed than the definition of the dependency path. It removes the requirement of the next egd on the path to have a dependency relationship with a governing term, but rather accepts any path that eventually leads to the root node. Such a path always exists during refinement and can be determined very efficiently. Since $\mathcal{SHOF}(\mathcal{D})$ lacks inverse roles, any egd

leading away from a node can be picked for a path that eventually leads back to the root node. Even when inverse roles are present, we simply pick an egd that leads into the direction of a predecessor node (with regard to the total order of nodes) to obtain a path in linear time.

The egd path is tailored specifically to the matching process: the egd path contains exactly those role axioms that are needed for some governing term $ax \in \kappa$ to appear in G.

Calculating the egd path for some axiom ax is possible by using the dependency map Dep. An implementation is shown with procedure collectEGDs (procedure 7.4). The implementation explicitly prefers direct generating egds. Since each node x typically has only a single generating dependency, the distinction is often irrelevant. However, when multiple egds are present, preferring the direct ancestor of a (governing) term remains closer to the graph's semantic interpretation.

Procedure 7.4: collectEGDs(G, ax, x_0)

Input: $G = (V, E, \mathcal{L}, \kappa, \mathsf{Dep}, \mathsf{Mer})$ an augmented completion graph
$ax = (x : C)$ a node axiom set entry from $\mathcal{L}$
$x_0 \in V$ a node from G, the root node
Output: edg: a set of axioms, the egd path of ax
begin

```
1   edg ← {ax};
2   (x_cur, C_cur) ← ax;
3   while x_cur ≠ x_0 do
        if ∃ y, r, B . (y : ∃ r . B) ↦ (x_cur, C_cur) ∈ Dep then
            /* We have a direct egd ancestor                    */
4           edg ← egd ∪ {(y : ∃ r . D)};
5           (x_cur, C_cur) ← ax;
        end
        else if ∃ y, r, C, B . (y : ∃ r . B) ↦ (x_cur, C) ∈ Dep then
            /* Pick an unrelated egd ancestor                   */
6           edg ← egd ∪ {(y : ∃ r . B)};
7           (x_cur, C_cur) ← ax;
        end
    end
8   return egd;
end
```

7.1.7. Logical Derivates

The reasoning process already leaves us with an explicit account of logical derivatives Considering such *child* axioms is also an option during token extraction. For example, given $\mathsf{TBox} = \{\mathsf{Bucher} \sqsubseteq \mathsf{Ashlar}\}$ and the axiom $(x : \mathsf{Bucher})$, we get a dependency to $(x : \mathsf{Ashlar})$. Collecting the child axioms is easily possible via the dependency map.

7.2 Similarity Measures for Completion Graphs

With the requirements from the previous section, we go on to design similarity measures for completion graph comparison. To enable evaluation, we first design a set of simple "baseline" similarity measurements. These baseline similarities will be subsequently refined, making more use of semantic information extracted during both refinement and mapping tableau application.

7.2.1. Baseline Similarities

As a baseline to compare against, we use a similarity measure without term filtering. filter (definition 7.1) is thus the identity function and we consequently extract phrases from all axioms $ax \in \mathcal{L}$. This also considers roles, because the refinement process introduces a corresponding existential term $(x : \exists\, r\,.\, C \in \mathcal{L}_s \cup \mathcal{L}_t)$ for every edge $r(x, y) \in E_s \cup E_t$ and extractToken (function 7.2) handles phrase extraction from role axioms (line 3).

The calls (source and target side) to extractToken leaves us with a two multisets of tokens Tk_s and Tk_t. To simplify subsequent algorithm descriptions, we use the following shortcut notations (see also appendix A.1)

- $\mathsf{Tk}_s(t)$ ($\mathsf{Tk}_t(t)$) is the number of occurrences of tk in Tk_s (Tk_t).
- The corpus Corp of Tk_s and Tk_t is the union of the supports of Tk_s and Tk_t: $\mathsf{Corp} = \mathsf{supp}(\mathsf{Tk}_s) \cup \mathsf{supp}(\mathsf{Tk}_t)$ (see appendix A.1).

- Additionally, let $\overrightarrow{\mathsf{Tk}_s}(\mathsf{Corp})$ be the vector of occurrences of strings from Tk_s with regard to the corpus, i.e.

$$\overrightarrow{\mathsf{Tk}_s}(\mathsf{Corp}) \equiv_{\mathrm{def}} (\mathsf{Tk}_s(\mathsf{tk}_0), \ldots, \mathsf{Tk}_s(\mathsf{tk}_n))$$

  for some arbitrary, but persistent ordering of the tokens in Corp.

The baseline similarity comes in three different variations, that differ in the way the similarity between Tk_s and Tk_t is calculated.

Flat Cosine Similarity

The first version uses basic cosine similarity (equation (7.1)) to compare token multisets.

$$\underset{\cos}{\mathsf{sim}}(\mathsf{Tk}_s, \mathsf{Tk}_t) \equiv_{\mathrm{def}} \frac{\overrightarrow{\mathsf{Tk}_s} \circ \overrightarrow{\mathsf{Tk}_t}}{\|\overrightarrow{\mathsf{Tk}_s}\|\|\overrightarrow{\mathsf{Tk}_t}\|} \tag{7.1}$$

Here, $\circ$ is the scalar product ($\overrightarrow{\mathsf{Tk}_s}$ are $\overrightarrow{\mathsf{Tk}_t}$ vectors). $\|\|$ is the Euclidean norm, i.e. the square root of the sum of squares. Note that we do not use token frequencies, but absolute token occurrences.

Flat TFIDF Cosine

The second version of flat cosine is more elaborate. In information retrieval, cosine similarity is typically found to give too much weight to frequent tokens. These tokens appear in many documents and thus have less information content than do rarer tokens. A possible remedy is to weigh the individual token frequencies by an additional factor. Again, the basic method from information retrieval involves a specific factor called *term frequency–inverse document frequency* (tfidf). To calculate the tfidf factor for a single input token $t \in T$, we need to consider all the completion graphs $\mathbb{G}$ in the current refinement graph. Note that the literature knows

many variations of the basic tfidf function. For this thesis, we will use *raw term frequency* and *unsmoothed, logarithmic inverse document frequency*, with a special case if $|\mathbb{G}| = 1$ (see definition 7.4).

Definition 7.4 Term Frequency–Inverse Document Frequency

Given a set of completion graphs $\mathbb{G} = \{G_0, G_1, ...\}$, and their respective extracted token multisets Tk_0, Tk_i, ..., the **term frequency** $\mathsf{tf}(\mathsf{tk}, \mathsf{Tk}_i, \mathbb{G})$ is

$$\mathsf{tf}(\mathsf{tk}, \mathsf{Tk}_i, \mathbb{G}) \equiv_{\text{def}} \frac{\mathsf{Tk}_i(t)}{\sum\limits_{G_j \in \mathbb{G}} (\mathsf{Tk}_j(\mathsf{tk}))}$$

the **inverse document frequency** $\mathsf{idf}(\mathsf{tk}, \mathbb{G})$ is

$$\mathsf{idf}(\mathsf{tk}, \mathbb{G}) \equiv_{\text{def}} \ln \frac{|\mathbb{G}|}{|\left\{\mathsf{Tk}_j(\mathsf{tk}) | G_j \in \mathbb{G}\right\}|}$$

and the **term frequency–inverse document frequency** tfidf is

$$\mathsf{tfidf}(\mathsf{tk}, \mathsf{Tk}_i, \mathbb{G}) \equiv_{\text{def}} \mathsf{tf}(\mathsf{tk}, \mathsf{Tk}_i, \mathbb{G}) \times \begin{cases} \mathsf{idf}(\mathsf{tk}, \mathbb{G}) & \text{if } |\mathbb{G}| > 1 \\ 1 & \text{if } |\mathbb{G}| = 1 \end{cases}$$

The special case $|\mathbb{G}| = 1$ is the norm when we perform known source or known target matching (see section 6.1.3). In the known source mode, $|\mathbb{G}_s| = 1$ and known target implies $|\mathbb{G}_t| = 1$. Without the special treatment, the value of the tfidf factor would be zero for $|\mathbb{G}| = 1$. The replacement value 1 has been chosen because it is the simplest constant factor.

We call this version of flat cosine similarity $\mathsf{sim}_{\cos,\mathrm{tfidf}}$.

Soft Cosine

The vector space model makes a hard distinction between individual tokens; even a single typographical error between two otherwise equivalent tokens causes a new vector axis to appear. For example, Ashlar and Aschlar are considered completely separate tokens. As a potential remedy for cases where this distinction is too hard, Sidorov2014 et al. [SGGAP14] propose a *softer* version of cosine similarity on the

basis of (lexicographical) similarity function between tokens. *Soft cosine* is specified in definition 7.5. Soft cosine needs a similarity function $\mathsf{sim}_{\mathsf{Tk}}$ to relate individual tokens.

Definition 7.5 Soft Cosine

Let Tk_s and Tk_t (forming a corpus Corp) be token multisets and let $\mathsf{sim}_T : \mathsf{Corp} \mapsto \mathbb{R}$ be a token similarity function.
The **soft cosine similarity** between Tk_s and Tk_t is then

$$\mathsf{sim}_{\mathrm{cosine,soft}}(\mathsf{Tk}_s, \mathsf{Tk}_t) \equiv_{\mathrm{def}} \frac{\sum_{\mathsf{tk}_s \in \mathsf{Tk}_s, \mathsf{tk}_t \in \mathsf{Tk}_t} \mathsf{sim}_{\mathsf{Tk}}(\mathsf{tk}_s, \mathsf{tk}_t) \times \mathsf{tk}_s \times \mathsf{tk}_t}{\sqrt{\sum_{\mathsf{tk}_{s,1} \in \mathsf{Tk}_s, \mathsf{tk}_{s,2} \in \mathsf{Tk}_s} \mathsf{sim}_{\mathsf{Tk}}(\mathsf{tk}_{s,1}, t_{s,2}) \times \mathsf{tk}_{s,1} \times \mathsf{tk}_{s,2}} \times \sqrt{\sum_{\mathsf{tk}_{t,1} \in \mathsf{Tk}_t, \mathsf{tk}_{t,2} \in \mathsf{Tk}_t} \mathsf{sim}_{\mathsf{Tk}}(\mathsf{tk}_{t,1}, t_{t,2}) \times \mathsf{tk}_{t,1} \times \mathsf{tk}_{t,2}}}$$

Soft cosine is a true generalization of cosine similarity. With

$$\mathsf{sim}_{\mathsf{Tk}}(\mathsf{tk}_s, \mathsf{tk}_t) = \begin{cases} 1 & \text{iff } \mathsf{tk}_s = \mathsf{tk}_t \\ 0 & \text{otherwise} \end{cases}$$

soft cosine similarity degenerates into cosine similarity (see [SGGAP14]), so it is a proper extension of the former. As is, we have a choice between a wide variety of *string distance* measurements to use. For example, Apache Lucene offers at least four different varieties of `StringDistance`[13] implementations.

Once again, for matters of simplicity, we use `LevensteinDistance`[14]. This is implemented by calculating the edit distance [Lev66] between the two input strings and normalizing the result by the length of the longer input string.

$$\mathsf{sim}_{\mathrm{edit}}(\mathsf{tk}_s, \mathsf{tk}_t) = \frac{\mathsf{levenshtein}(\mathsf{tk}_s, \mathsf{tk}_t)}{\mathsf{max}(\mathsf{len}(\mathsf{tk}_s), \mathsf{len}(\mathsf{tk}_t))} \tag{7.2}$$

Calculating soft cosine is more expensive (linear vs. quadratic time), but makes room for non-binary relationships between tokens.

[13] interface `org.apache.lucene.search.spell.StringInstance`
[14] class `org.apache.lucene.search.spell.LevensteinDistance`

Table 7.3. Variations of Governing Term Cosine Similarity

Phrase Similarity	flat cosine cosine-tfidf soft cosine
EGD-Path	follow/don't follow
Children	collect/don't collect

We now have three different versions of the basic "flat" similarity measure: $\mathsf{sim}_{\mathrm{cos}}$, $\mathsf{sim}_{\mathrm{cos,tfidf}}$, and $\mathsf{sim}_{\mathrm{cosine,soft}}$.

7.2.2. Governing Term Cosine Similarity

The first variation we introduce involves term filtering. Instead of evaluating every axiom from $\mathcal{L}$, we consider only governing terms κ. The result is *governing term flat cosine*, as we still use unstructured evaluation of terms (flat), but now start with the governing terms κ.

This change adds to the list of parameter choices. Both the egd-path (section 7.1.6) and logical derivates section 7.1.7 become meaningful concepts, leaving us with 24 different variations of governing term cosine (table 7.3).

7.2.3. Clustered Cosine Similarity

Our next variation of a similarity involves structuring the set of input axioms following semantic information from the refinement process. Remember back from chapter 6, that we consider each refinement step as the introduction of a single, new semantic feature. Given this assumption, it seems natural to cluster the filtered axioms from a completion graph with regard to the refinement step they were introduced in.

Consider fig. 6.1b way back from the introductory example. Here, $(x : GF)$ was initially present (and it is also present for all completion graphs with the same initial alignment). In the next refinement step, we introduced $(x : \exists\, fW\,.\,\top)$. The

Table 7.4. Example Axiom-Clustered Alignment

1.	$(x : \mathsf{GF})$	$(x : \mathsf{GF})$
2.		$\{(x : \exists\, \mathsf{fW} \,.\, \top), (y : \mathsf{FM})\}$
3.	$(x : \exists\, \mathsf{uM} \,.\, \top), \big(y : \{\underline{\mathsf{plaster}}\}\big)$	$(y : \mathsf{P})$

reasoner created the successor node y and introduced $(y : FM)$ as a logical consequence. $(x : \exists\, fW \,.\, \top)$ and $(y : FM)$ form a logical unit, the former being an egd of the latter. The final step introduces either $(y : S)$, $(y : P)$, $(y : B)$, or $(y : L)$.

The result of the process are three *axiom clusters*, one for each refinement step. If we look at the source side of the mapping (fig. 6.1a), we see a similar picture: the initial axiom is the same ($(x : \mathsf{GF})$). There is only one refinement step and the second axiom sets are $\big\{(x : \exists\, \mathsf{uM} \,.\, \top), \big(y : \{\underline{\mathsf{plaster}}\}\big)\big\}$ $\{(x : \exists\, \mathsf{uM} \,.\, \top), (y : \{\underline{\mathsf{brick}}\})\}$, ...

The resulting *clusters* of governing terms are shown in table 7.4.

Instead of viewing the (source and target) completion graphs as two unstructured sets of axioms, we have used information from the refinement process and the reasoner to split the axiom sets into semantic sub-clusters. These sub-clusters can now be compared individually.

To calculate the similarity between these sets of clusters, we repeatedly pick the best match between source and target until at least one side of runs out of axiom clusters. In the example, we thus first match $(x : \mathsf{GF})$ against $(x : \mathsf{GF})$, obtaining the assignment $(1, 1)$. The rest of the correspondences are not so clear, however. Is it better to align $(x : \exists\, \mathsf{uM} \,.\, \top)$, $\big(y : \{\underline{\mathsf{plaster}}\}\big)$ with $\{(x : \exists\, \mathsf{fW} \,.\, \top), (y : \mathsf{FM})\}$ or with $(y : \mathsf{P})$? What we see here, is an artefact of the refinement process. In the target ontology, the specification of a concrete material is optional, while the source ontology explicitly forces a value for the material datatype.

It is also possible to interpret this as an artefact of expressive power of the underlying logic, as both OWL and OWL2 do not have the ability to express *abstract classes* for which no pure instances may exist.

Table 7.5. Example Expanded Axiom-Clustered Alignment

1.	$(x : \mathsf{GF})$	$(x : \mathsf{GF})$
2.		$(x : \exists\, \mathsf{fW} \,.\, \top)$, $(y : \mathsf{FM})$
3.	$(x : \exists\, \mathsf{uM} \,.\, \top)$, $\big(y : \{\underline{\mathsf{plaster}}\}\big)$	$(x : \exists\, \mathsf{uM} \,.\, \top)$, $(y : \mathsf{P})$

The result of the artefact is that on the source side, there is only a single semantic feature, while on the target side we require two steps to obtain the same level of refinement. A partial remedy to the problem is provided by applying the results from section 7.1.6. Applying collectEGDs to $(y : \mathsf{P})$ *after* cluster extraction results in the axiom set $\{(x : \exists\, \mathsf{fW} \,.\, \top)\,, (y : \mathsf{P})\}$.

The resulting expanded alignment is shown in table 7.5. As is possible to see, the source side in line 2 is empty. Target cluster 2 ($\{(x : \exists\, \mathsf{fW} \,.\, \top)\,, (y : \mathsf{FM})\}$) is left *dangling*, i.e. it has no source counterpart to match against.

The resulting similarities are summed up, but not normalized. This gives priority to longer matches where more clusters are aligned with each other. We call this similarity measure *clustered cosine similarity*.

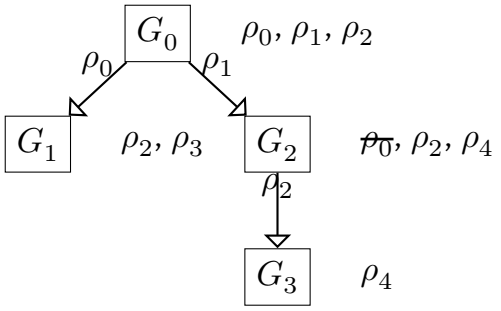
7.2.4. Dependency Cluster Similarity

In the previous section, we have deliberately ignored unmatched axiom clusters. It should be easy to see that this seems unreliable: a completion graph with many unmatched axiom clusters compares the same as one with only matched axiom clusters, as long as their core of matched axiom clusters is the same. Upon closer observation, however, some of the unmatched axiom are clearly relevant. In section 7.2.3 (table 7.5) $2 : \{(x : \exists\, \mathsf{fW} \,.\, \top)\,, (y : \mathsf{FM})\}$ cannot be removed, because it is necessary for $3 : \{(x : \exists\, \mathsf{uM} \,.\, \top)\,, (y : \mathsf{P})\}$ to appear. The existence of 2 is *justified*, because 3 has a good match on the source side.

Colloquially, an axiom cluster is a *justified* if it is

- matched with an axiom cluster on the other side of the alignment, or
- required for another, justified axiom.

Figure 7.2. Generating Refinement Rules



While the former is clear enough, the latter requires additional formalization. To do this, we need the concept of the *generating refinement rule*. Remember back from section 6.2.4 that rules are either applicable or not applicable to a particular completion graph. Add to this the abstract refinement graph in fig. 7.2.

Here, ρ_0, ρ_1, and ρ_2 are initially applicable. After applying ρ_0, ρ_3 becomes applicable in G_1. After applying ρ_1, ρ_4 is newly applicable in G_2. This means that ρ_0 is the generating refinement rule of ρ_3 and ρ_1 is the generating refinement rule of ρ_4.

Note that the concept is different from the order of rule application. ρ_2 is only applied after ρ_1, but ρ_1 is *not* the generating refinement rule for ρ_2. Because ρ_2 was already applicable at the start, it has no generating rule.

Another way of viewing the situation is that one has to apply ρ_1 to obtain ρ_4, but it is not necessary to apply ρ_1 to obtain ρ_2. Relating this concept to axiom justification is straightforward: an axiom $(y : D)$ is justified by some other axiom $(x : C)$ if $(y : D)$ was introduced via the refinement rule ρ_y and $(y : D)$ was

- initially present, or
- introduced as a result of or before the generating refinement rule ρ_x of ρ_y.

Again, colloquially, $(x : C)$ has to exist before $(y : D)$ can be introduced. Justification is transitive, i.e. if an axiom is justified, it also justifies all its ancestors.

Figure 7.3. Dependency Cluster Constraint Matrix

$$M = \begin{pmatrix} \mathsf{sim}_{0,0} & \mathsf{sim}_{0,1} & \ldots & \mathsf{sim}_{0,m} & 0 & -1 \\ \mathsf{sim}_{1,0} & \mathsf{sim}_{1,1} & \ldots & \mathsf{sim}_{1,m} & 0 & -1 \\ \vdots & \vdots & \vdots & \vdots & & \\ \mathsf{sim}_{n,0} & \mathsf{sim}_{n,1} & \ldots & \mathsf{sim}_{n,m} & 0 & -1 \\ 0 & 0 & \ldots & 0 & 0 & 0 \\ -1 & -1 & \ldots & -1 & 0 & 0 \end{pmatrix}$$

Extending clustered cosine similarity with this additional knowledge is possible, but requires introduction of a similarity threshold. Because $\mathsf{sim}_{\text{cosine}} = 0$ only for completely disjoint sets of axioms, every axiom can usually be matched with every other axiom as axioms usually share at least one token. The cut-off threshold is set to

$$0.05 * \underset{i,j}{\mathsf{avg}}(\mathsf{sim}_{i,j})$$

This means, we consider only the 5% worst axiom similarities relative to the current set of corresponding axiom clusters to be non-pairable. The constant 0.05 has been determined empirically. Both increasing as well as decreasing the value showed adverse effects for matcher runs. Note, however that the cut-off parameter maybe specific to a particular alignment task.

To implement dependency cluster matching, we need to formulate a constraint system. A (relatively) simple way of doing this is to write it down in matrix form as shown in fig. 7.3. Assume that there are m source axiom clusters and n target axiom clusters. $\mathsf{sim}_{i,j}$ is the truncated similarity between $T_{s,i}$ and $T_{t,j}$, respectively. Each position in the matrix gets assigned a boolean variable $x_{i,j} \in \{0, 1\}$. The $x_{i,j}$ form a second matrix X.

The rationale for this form is as follows:

- The value domain for all $x_{i,j}$ is $\{0, 1\}$, i.e. we are using mixed integer linear programming (MILP, [Wol08]).

- For $0 \leq i \leq m$ and $0 \leq j \leq n$, setting $x_{i,j} = 1$ assigns the axiom cluster $T_{s,i}$ to the axiom cluster $T_{t,j}$.
- Setting any of $x_{i,n+1}$ means that $T_{s,i}$ is unmatched, but justified.
- Setting any of $x_{m+1,j}$ means that $T_{t,j}$ is unmatched, but justified.
- Setting any of $x_{i,n+2}$ means that $T_{s,i}$ is unmatched *and* unjustified.

  As we can see, unjustified axioms are strongly discouraged by assigning them a negative weight
- Setting any of $x_{m+2,j}$ means that $T_{t,j}$ is unmatched *and* unjustified.
- $x_{m+1,n+1}$, $x_{m+2,n+1}$, $x_{m+1,n+2}$, and $x_{m+1,n+2}$ purely exist to make the matrix square and have no special meaning. They should always be zero.

To make this interpretation work, we must apply additional constraints to X:

target term single assigment For each $i \in \{0, \dots, m\}$, $\sum_{j \in \{0,\dots,n+2\}} x_{i,j} = 1$

Every target term must be assigned exactly once.

source term single assigment For each $j \in \{0, \dots, n\}$, $\sum_{j \in \{0,\dots,m+2\}} x_{i,j} = 1$

Every source term must be assigned exactly once.

source justified constraint For each $i \in \{0, \dots, m\}$, $x_{i,n+1}$ must be smaller or equal to the sum of all the assignments of all the justifiers of $T_{s,i}$.

For example if $T_{t,j}$ is the only justifier of $T_{s,i}$, then

$$x_{i,n+1} \leq \sum_{k \in \{0,\dots,m+1\}} x_{k,j}$$

If there are no justifiers, $x_{i,n+1}$ is forced to zero unless the corresponding term was already present in the initial graph G_0. This ensures that a source term can only move to the justified state if it actually has a justifier.

target justified constraint Vice versa, for each $j \in \{0, \dots, n\}$, $x_{m+1,j}$ must be smaller or equal to the sum of all the assignments of all the justifiers of $T_{t,j}$.

Table 7.6. Example Axiom Cluster Similarity Matrix

		t_0 $(x : \mathsf{GF})$	t_1 $(x : \exists\, \mathsf{fW} \,.\, \top)$, $(y : \mathsf{FM})$	t_2 $(x : \exists\, \mathsf{uM} \,.\, \top)$, $(y : \mathsf{P})$
s_0	$(x : \mathsf{GF})$	1.00	0.01	0.01
s_1	$(x : \exists\, \mathsf{uM} \,.\, \top)$, $(y : \{\underline{\mathsf{plaster}}\})$	0.01	0.25	0.75

Once again, if $T_{s,i}$ is the only justifier of $T_{j,j}$, then

$$x_{m+1,j} \leq \sum_{k \in \{0,\ldots,n+1\}} x_{i,k}$$

If there are no justifiers, $x_{m+1,j}$ is forced to zero unless the corresponding term was present in the initial graph G_0.

Using this constraint system we want to find the best possible assignment of terms that maximizes the overall sum of similarities. This can be done by using a mixed integer linear constraint optimizer, solving the equation

$$\mathsf{sim}_{\mathrm{depcluster}} = \max \sum M \circ X \tag{7.3}$$

As indicated by the function name, we will use this value as the similarity function for *dependency cluster similarity*, $\mathsf{sim}_{\mathrm{depcluster}}$.

As an example, consider the term cluster from table 7.5 together with the axiom set similarities given in table 7.6.

Combining these yields the similarity matrix M from equation (7.4).

$$M = \begin{pmatrix} 1.00 & 0.01 & 0.01 & 0 & -1 \\ 0.01 & 0.25 & 0.75 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & 0 & 0 \end{pmatrix} \tag{7.4}$$

The variable matrix X (equation (7.5)) is of the same dimension. As indicated, the filler variables $x_{2,3}$ $x_{3,3}$, $x_{2,4}$, and $x_{3,4}$ are forced to zero.

$$X = \begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} = 0 & x_{2,4} = 0 \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} = 0 & x_{3,4} = 0 \end{pmatrix} \tag{7.5}$$

The constraints for the matrix are given below.

- Since they are the last terms in the refinement, neither s_1 nor t_2 have a justifier. Both $x_{1,3}$ and $x_{2,2}$ are thus forced to zero, i.e. may not be assigned.

$$x_{1,3} = 0 \tag{7.6}$$
$$x_{2,2} = 0 \tag{7.7}$$

- Single term assignment yields that the row sum for the first two rows and that the column sum for the first three columns is one.

$$x_{0,0} + x_{0,1} + x_{0,2} + x_{0,3} + x_{0,4} = 1 \tag{7.8}$$
$$x_{1,0} + x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1 \tag{7.9}$$

$$x_{0,0} + x_{1,0} + x_{2,0} + x_{3,0} = 1 \tag{7.10}$$
$$x_{0,1} + x_{1,1} + x_{2,1} + x_{3,1} = 1 \tag{7.11}$$
$$x_{0,2} + x_{1,2} + x_{2,2} + x_{3,2} = 1 \tag{7.12}$$

- s_0 is needed for s_1, i.e. s_1 justifies s_0. But since s_0 was initially present, we do not encode this constraint.

- t_0 is needed for t_1, but since t_0 was initially present, we do not encode a constraint for it. However, t_1 was not initially present and it is needed for t_2, hence we obtain equation 7.13.

$$x_{2,1} \leq x_{0,2} + x_{1,2} + x_{2,2} \tag{7.13}$$

Optimizing this constraint system to maximize $\Sigma M \circ X$, we obtain

$$\begin{pmatrix} x_{0,0}=1 & x_{0,1}=0 & x_{0,2}=0: & x_{0,3}=0 & x_{0,4}=0 \\ x_{1,0}=0 & x_{1,1}=0 & x_{1,2}=1 & x_{1,3}=0 & x_{1,4}=0 \\ x_{2,0}=0 & x_{2,1}=1 & x_{2,2}=0 & x_{2,3}=0 & x_{2,4}=0 \\ x_{3,0}=0 & x_{3,1}=0 & x_{3,2}=0 & x_{3,3}=0 & x_{3,4}=0 \end{pmatrix} \tag{7.14}$$

s_0 is mapped to t_0, s_1 is mapped to t_2 and t_1 is dangling, but justified by t_2. This is reflected in the assignment: it was only possible to assign $x_{2,1} = 1$, because $x_{1,2}$ was already non-zero (as per constraint equation (7.13)).

The implementation makes use of the Choco [JRL08] constraint solver.

Chapter Summary

In this chapter, we have designed multiple similarity measures to compare DL completion graphs as obtained from the refinement process described in section 6.1.

Starting from a relatively simple flat similarity measure that simply views a completion graph as a unstructured set of logical axioms (in $\mathcal{L}$), we iteratively enhanced this basic measurement using semantic information from both refinement and reasoning steps. Herein, we also made use of non-standard reasoning services: The dependency map Dep and the governing term set κ from the mapping tableau (section 5.2) provided us with additional semantic information.

The result is clustered cosine similarity, where we separate axioms from κ (possibly augmented from the egd-path and with logical derivates) into distinct clusters, which we match against their corresponding counterparts individually. The new similarity measure also a a full set of new parameters (term collection) to evaluate.

In a third version of a similarity measure, we incorporate the relationship between refinement rules in the form of *generating refinement rules* (figure 7.2). This lead us to the concept of axiom cluster *justification* and from there to a new *dependency cluster* (depcluster) similarity measure.

All three of similarity measure, flat cosine, clustered cosine and depcluster similarity will be evaluated empirically in the next, final section of this thesis.

SECTION 8.0

Evaluation and Conclusion

The final chapter of this thesis focusses on the empirical evaluation, in particular on the evaluation of the three completion graph similarities flat cosine, clustered cosine, and depcluster. The effects of matcher parameter choices are also presented.

We start with a description of the basic evaluation tool set, most importantly how the correctness of generated mapping rules is evaluated. We then describe the ontologies used during the evaluation and the design of the evaluation process as a whole.

The chapter concludes with a summary of the observed results and gives an outlook on future improvements.

8.1 Comparing Mapping Rules

To compare conjunctive queries, simple syntactic evaluation is insufficient. The two terms $A(?x)$ and $A(?y)$ are syntactically different, but semantically equivalent when both $?x$ and $?y$ are free variables.

In addition, background knowledge contained in the TBox, can make two syntactically very different rules semantically equivalent. For example, consider

$\mathsf{TBox} = \{A \sqsubseteq B, B \sqsubseteq A\}$.

In this case, $B(?x)$ and $A(?y)$ are semantically equivalent, since $A \equiv B$. A similar case holds e.g. given $\mathsf{RBox} = \{r \sqsubset q, q \sqsubset r\}$. Now, $r(?x, ?y)$ and $q(?w, ?z)$ are semantically equivalent.

We use augmentation of the rule terms to solve this problem for our use case. The expansion is incomplete in general, but is sufficient for the mapping use case.

When two rule terms Φ_r, Φ_m are compared, we augment both Φ_r and Φ_m with selected terms from the other, maintaining interpretation.

In particular

- If $A_r(?x)$ is a class term present in Φ_r, and $B_m(?y)$ is a class term present in Φ_m, and $\mathsf{TBox} \models A_r \sqsubseteq B_m$, add $B_m(?x)$ to Φ_r.

  The same procedure is applied in the reverse direction.

- Similarly, if $q_r(?x, ?y) \in \Phi_r$, $s_m(?w, ?z) \in \Phi_m$, and $\mathsf{RBox} \models q_r \sqsubseteq s_m$, add $s_m(?x, ?y)$ to Φ_r.

  Also again, the same procedure is applied in the reverse direction.

To validate generated rules against the reference rule sets, we once more make use of the Choco [JRL08] constraint solver on the augmented terms.

This leaves us with a binary classification between rules that are equal to existing rules. To introduce at least some measure of fuzziness, we also consider *correct* rules.

A rule is correct, if does not transfer invalid information. This is the case, when

- the rule head is equal or more specific than the rule head of a reference rule, or
- the rule body is equal or less specific than the rule body of a reference rule.

For example, if the reference rule is $A(?x), r(?x, ?y), B(?y) \mapsto C'(?x), D'(?x)$, the following are examples for correct rules:

$$A(?x), r(?x, ?y), B(?y), C(?y) \mapsto \quad C'(?x), D'(?x), \tag{8.1}$$

$$A(?x), r(?x, ?y), B(?y) \mapsto \quad C'(?x) \tag{8.2}$$

$$A(?x), r(?x, ?y), B(?y) \mapsto \quad D'(?x) \tag{8.3}$$

$$A(?x), r(?x, ?y), B(?y), C(?y) \mapsto \quad C'(?x) \tag{8.4}$$

A correct rule is semantically justified, but does not transfer as much information as possible from source to target. If the head of a rule is over-refined, it excludes some sub-graphs from the source side that could be mapped. If the body of a rule is under-refined, it generates a smaller target sub-graph than what would be optimal.

In the known-target case, only the body part is relevant, because the rule head is derived from a user-supplied query. If known-target matching produces many correct rules for some similarity computation, this is an indicator that the calculation has a tendency to under-refine rules.

In the known-source case, only the head part is relevant, because the rule body is derived from a user-supplied query. Conversely, a large number of correct rules in the known-source case indicates a tendency to over-refine rules.

8.2 Evaluation Design

Evaluation is performed using two ontologies O_1 and O_2.

O_1 is the original built heritage ontology as extracted from the University of Bamberg's "Mobile Mapping System" software. This is the actual ontology used for documentation of damage, restoration, and repair work for the Bamberg Cathedral.

O_2 is an evolved version of O_1. It represents the same information set, but uses more refined modelling methods not available in the mapping software itself. The modelling in O_2 is typically more complex and more extensible, but also in parts more restricted, making strong use of TBox constraints whenever possible. The logical constraints in O_1 are sometimes too lax, allowing for consistent but nonsensical models, O_2 is much more restricted while at the same time providing well-defined extension paths.

Mapping from O_1 to O_2 requires complex alignments. The results shown in table 3.1 have already demonstrated that simple alignments are not only insufficient but that the need for complex alignment reduces alignment quality obtained from simple matchers.

The evaluation is performed in two steps that evaluate and highlight different aspects of the refinement process.

1. At first, I evaluate the suitability of the refinement process for interactive mapping refinement. Interactive mapping is performed using the tool described in section 6.3.
2. The second step evaluates the automated alignment provided by the similarity measures developed in developed in chapter 7.

Evaluation of Interactive Refinement

Automated refinement lends itself well to quantitative evaluation (see below), but the situation is not so simple for interactive refinement. Measuring the user experience is problematic. Hence, usability will only be evaluated anecdotally. This is done by reviewing the experience during the establishment of the reference rulesets.

In particular, the following potential problem areas will be observed:

Incompleteness If the iterative matching system is not capable of deriving a refinement rule, it is incomplete with regard to the use case.

Information Scarcity This happens when the user is not presented with sufficient information to make a refinement decision.

Information Overload This situation occurs, when a user is presented with too much information at once, i.e when the filtering or structuring of information provided by the matching system is insufficient.

Evaluation of the Automated Alignment

The result of the first step is a (complex) reference alignment A_1. A_1 is complete, i.e. it transfers as much information as possible from O_1 to O_2.

I use the reference alignment A_1 (and a variation A_2 of it) to evaluate the completion graph similarity measures developed in chapter 7.

The auto-generated alignments will be compared against the reference alignments. I will measure the precision (number of correct alignments). Additionally, we will also measure the number of *correct* alignments as explained in section 8.1.

Reference Alignments

In section 6.3 I showcased a tool for interactive refinement of initial correspondences. In the first step of the evaluation, the tool will be used to derive the reference alignment from O_1 to O_2.

The initial, simple alignment between O_1 and O_2 is generated using AgreementMaker [CAS09]. Subsequently, we use the interactive refinement tool (section 6.3) to generate a reference alignment A_1. This alignment is then inspected for completeness, that is we check if the reference alignment is *complete* in the sense that it transfers as much information as possible from O_1 to O_2.

For further evaluation, we use two different reference alignments:

Alignment 1 As already mentioned, the first alignment A_1 contains all maximally refined correspondences between O_1 and O_2.

This means that if both

$$A(?x), r(?x, ?y), B(?y) \mapsto C(?x), q(?x, ?z), D(?z)$$

and

$$A(?x) \mapsto C(?x)$$

are valid correspondences, only the most specific (section 8.2) is represented.

A_1 contains 88 complex mapping rules.

Alignment 2 on the other hand is cut-off version of A_1. Some of the correspondences are less refined. For example, the correspondence from section 8.2 could be reduced to

$$A(?x), r(?x, ?y), \top(?y) \mapsto C(?x), q(?x, ?z), \top(?z)$$

Table 8.1. Matcher Parameters for the Evaluation

parameter	possible values	reference
reference terms	ref_s, ref_t	section 6.1
matcher mode	(known-)target, (known-)source	section 6.1.3
phrase extraction	stemmer, stemmer-decomposer, normalizer	section 7.1.4
dependent axiom collection	none, egd, children, egd-children	section 7.1.5
string similarity	cosine, cosine-tfidf, softcosine	section 7.2.1
completion graph similarity	flat, govterm-flat, govterm-clustered, depcluster	section 7.2
reference alignment	A_1, A_2	section 8.2

Because of the reduction, Alignment 2 contains fewer mapping rules. Only 64 complex correspondences are represented.

A_2 has been explicitly designed to simulate selective extraction of limited ontology/knowledge base fragments as outlined in section 6.1.

The evaluated refinement processes will be performed in two directions (see section 6.1). In the first set of evaluation runs, we will fix the target side (by providing ref_t) in line with the preservation scenario that is the basis for this thesis. In the second set of evaluation runs, the source side (ref_s) will be fixed, meaning that we search target ontology fragments for a set of known source ontology fragments.

Any alignment obtained from the automatic refinement process will be compared against the reference alignment.

Table 8.1 gives an overview of the different matcher parameters and their possible values. In total, there are 576 possible combinations, but not all combinations of matcher parameters are sensible. For example, *flat* cosine similarity cannot be combined with any of the term cluster expansion methods (egd, children, egd-children) as the flat term model already uses *all* terms from the completion graph.

8.3 Evaluation Results

Tables with the raw evaluation results can be found in appendix A.3.

8.3.1. Interactive Refinement

The interactive refinement process manifests itself as very useful.

The reference alignment generated by the interactive method is complete with regard to the limitations of the process itself (i.e. no value mapping, no multi-variable mapping).

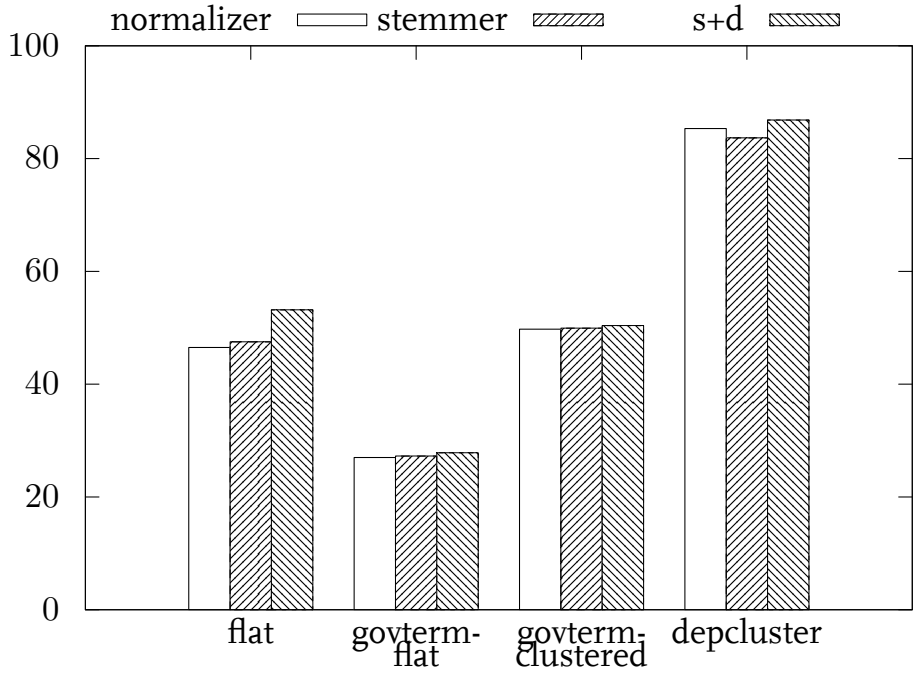
The context-sensitive presentation of the refinement steps was useful. Visualizing the attachment points of an alignment made it possible to understand subordinate relationships. For example, when the refinement Bucher is presented attach to Material it is clear from the displayed graph structure that a material is introduced as long as there is at least limited domain knowledge.

If a refinement on one side required multiple refinements on the other side, the interactive process was sometimes not straightforward: for example, when the source governing terms are Vierung and Bucher, but the target requires the introduction of hasMaterial before the concrete stone type (Bucher) can be introduced, this often requires some experimentation by the user. This problem was worsened by the fact that the system currently does not provide undo functionality.

There was never information overload during the process. At any point in time, there where usually only up to five applicable refinement rules and never more than three refinement rules per graph node. The context sensitive presentation is very helpful in this regard.

In general the interactive tool allows a user to quickly establish a set of complex refinements in a matter of minutes, which is a huge improvement over a non-assisted process.

Figure 8.1. Effects of Extractor Parameters



Score is averaged over all performed matcher runs. Missing entries indicate configurations that were not evaluated.

8.3.2. Effects of Lexical Extractor Parameters

The remainder of the evaluation concerns itself with the fully automatic derivation of complex alignments. Results here are more mixed than for the assisted case. The automatic derivation is also more suited to quantitative evaluation

We first study the effects of the string similarity measure and lexical extraction setup on alignment quality.

Figure 8.1 shows the averaged score for each similarity measure depending on the choice of extractor parameter. There is a slight drop-off for depcluster in the s+d (stemmer-decomposer) variant. This affects only 2 (A_1) or 1 (A_2) correspondences and is probably only an artefact from the limited scope of the evaluation.

In general, configurations which use more elaborate lexical pre-processing perform better.

8.3.3. Effects of Axiom Set Similarity

Comparing the effects of the string set similarity measure has interesting results. Traditional cosine similarity yields the best overall results.

The added computational complexity of soft cosine results in little benefit. This is most likely, because the evaluation ontologies use controlled jargon and contain too few lexical errors as to benefit from the added fuzzing of soft-cosine. It is important to note, however, that the negative performance effect of soft-cosine is relatively small. Also, the added fuzziness of soft cosine does not affect the quality of the results.

The comparison (fig. 8.2) shows cosine-tfidf score lower on average for all completion graph similarity measures, which is astounding. When considering the raw evaluation results (see appendix A.3), matching the first dataset A_1 with flat and govterm-flat needs to be considered a total failure, with only a single correct correspondence returned ($Prec = 1.1\%$). A similar, if somewhat better result can be observed when matching A_2, with a precision of up to 27%. Results are slightly better for govterm-clustered and depcluster similarities, but basic cosine and soft-cosine still perform better.

8.3.4. Effects of Axiom Collection

Figure 8.3 shows the effects of the axiom collection steps, i.e. following the egd-path and collecting logical children. Both govterm-flat and govterm-clustered perform significantly better in all cases, when both collection methods (egd+children) are used. The result is strong enough that evaluation of depcluster was limited to using only the egd+children parameter set.

8.3.5. Comparison Of Completion Graph Similarity Measures

Figure 8.4 shows the best results by completion graph similarity measure. As noted in section 8.3.3, cosine-tfidf was found to produce erratic results. To measure overall quality, TFIDF scores have thus been eliminated from the analysis.

Figure 8.2. Effects of Term Set Measure

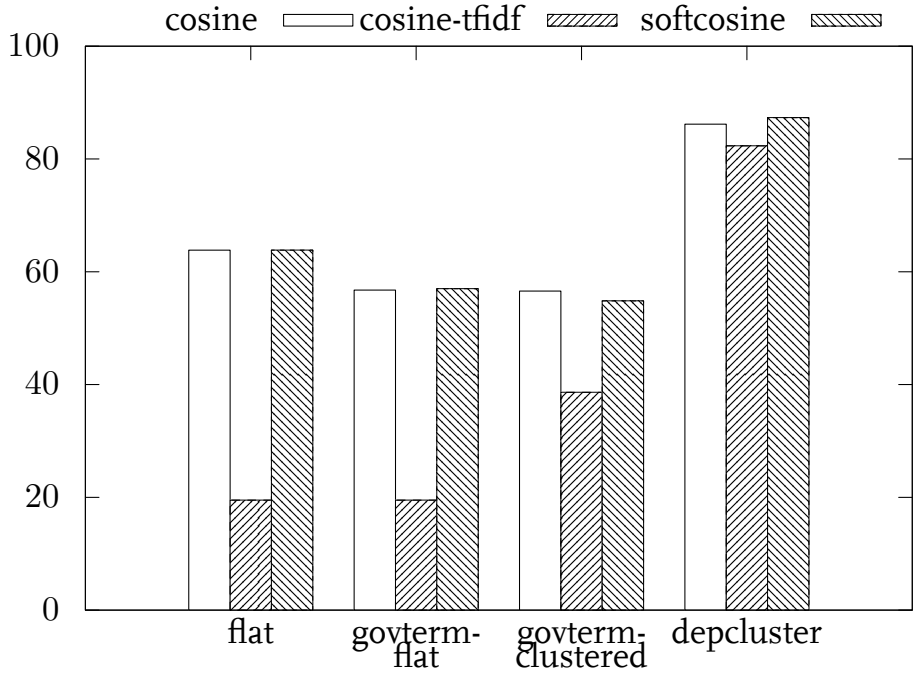
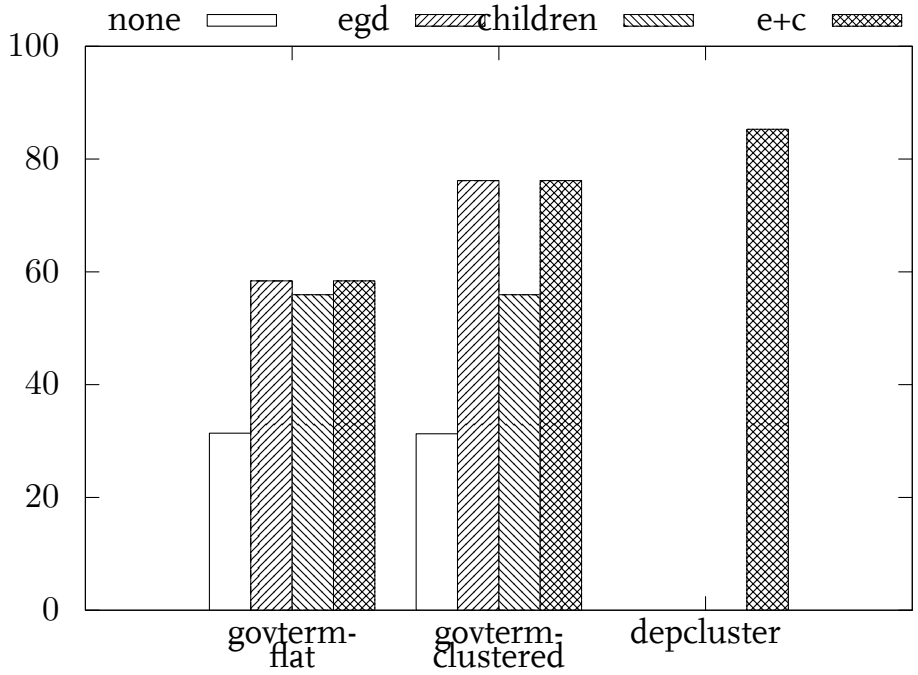


Figure 8.3. Effects of Dependent Term Collection



Score is averaged over all performed matcher runs. Missing entries indicate configurations that were not evaluated.

Up so far, we have only evaluated matcher runs that use a fixed target (known-target). For comparison of the best match obtainable from each completion graph similarity, we also consider matcher runs with a fixed source (known-source).

Known-target When matching the full (A_1) alignment, there is a constant improvement from the simple to the more elaborate similarity measures. Govterm-flat performs better than flat, govterm-clustered better than govterm-flat, with depcluster similarity yielding the best results.

When the reduced reference alignment (A_2) is evaluated, however, the results become more varied. After the initial improvement from flat to govterm-flat, there is now a slight, but noticeable drop-off in precision.

In fact the matcher quality decreases from flat cosine to govterm-flat, and then to to govterm-clustered. In absolute numbers, however, the difference is only two refinement rules (61 vs. 60 vs 59).

Known-Source The best-match evaluation for known-source refinements show a quite mixed picture. In the full refinement variant (A_1), govterm-flat performs significantly better than flat cosine. With a precision of only up to 6% for A_1 and 27% for A_2, govterm-clustered is unsuited for the known-source use case.

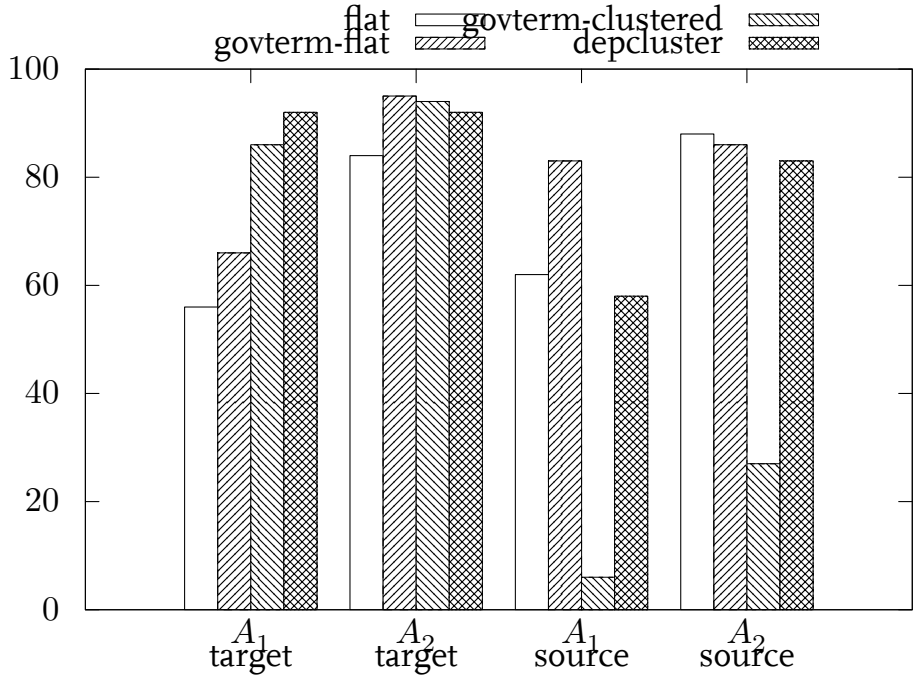
The problems are only partially resolved by the upgrade to depcluster similarity. Flat cosine and govterm-flat show the best precision for the known-source case.

8.4 Discussion

8.4.1. Interactive Refinement

While the overall *feel* of the interactive refinement tool seems good, a more thorough evaluation would have been necessary to provide dependable results. In particular, a study with multiple users with varying domain expertise would have been useful.

Figure 8.4. Best Results by Completion Graph Similarity



Shown is best obtain precision, limited to non-TFIDF scores (see discussion).

However, even as is, it can be stated that interactive refinement is very useful. The current state of the refinement tool was completely sufficient to establish the reference alignment and there are clear paths for further improvement (see section 8.6).

8.4.2. Automatic Refinement

Overall results from the evaluation of automatic refinement are mixed (fig. 8.4). The maximally refined rule scenario (A_1) benefits from all of the newly developed methods. In the cut-off scenario (A_2), the situation is different: here, there is a strong gap between the totally unstructured flat cosine and the structured methods (starting with govterm-flat). The added effort for govterm-clustered and depcluster, however, does not offer significant improvement for the cut-off scenario (A_2).

For the known-source variant, the picture is more varied: here, only govterm-flat improves or is at least en par with basic flat cosine similarity. Govterm-clustered similarity strongly underperforms and depcluster similarity remains slightly below the performance of flat cosine similarity.

These results can be explained by the structure of the ontologies involved and also shines light on some additional aspects:

Lexical Preprocessing is Beneficial

It shows across all evaluation runs that using the most elaborate term extraction pipeline delivers the best overall matching results.

The Matching Pipeline is an Integrated Process

The bad performance of TFIDF cosine for all but the depcluster matching in its A_1 incarnation is an indication that stacking well-known methods in a pipeline is not sufficient. Interaction between the different pipeline operations are more subtle.

TFIDF as implemented seems to be a bad fit for the matching scenario. There are relatively few documents that are also very short with only few terms. The set of completion graphs extracted from the refinement process is probably also not a representative set, which further skews the relevance calculation.

A similar problem is observed when analysing the causes for the failing of govterm-clustered in the known-source variant. Govterm-clustered cannot distinguish candidates that differ only in dangling clusters (those not matched to an opposite cluster). This combines badly with the fact the matching algorithm prefers longer matches in the case of a tie from the underlying similarity computation.

Semantic Information is Beneficial

Governing-term based matching in general works as intended and improves matching result as shown by the performance of govterm-flat similarity.

When clustering the governing terms before comparison, results are not so encouraging. While an improvement is noticeable when fixing the target side of alignment rules (known-target), govterm-clustered similarity seems very much unsuited to the evaluation case. Depcluster similarity partially remedies this problem, but still lags behind in the known-source case.

Matching Direction is Important

When inverting the matching direction, from known-target to known-source, the expected result was that overall quality of generated correspondences is lower, because the target (and thus presumed future) ontology is more complex than the source ontology. Hence fixing the target ontology reduces the search space and thus the alignment quality.

This argument is quite strongly supported by the execution time of the alignment search. In the known-source case, running times are over five times as long as their known-target counterparts. There are more potential target candidates than there are potential source candidates, i.e. for known-source the search space is much larger.

The expectation was fulfilled by all employed similarity measures.

However, there is also an artefact. While the known-target variant shows either an improvement or en-par performance when moving from govterm-flat to govterm-clustered to depcluster, the known source variant has govterm-flat as the best similarity measure with depcluster lagging slightly behind.

Govterm-clustered is seems prone to over-refining. This can be observed in tables A.4 and A.5. Govterm-clustered has the highest number of correct rules with over-refined source queries of all the similarity measures. This is because a dangling cluster has no negative impact on the calculated similarity, since only the best matches are picked, with leftover clustered being ignored. Consequently, govterm-clustered does not have the ability to distinguish between Balkenlager$(?x)$, Partonomie$(?x)$ (bar stock, partonomy) and Balkenlager$(?x)$.

This property of govterm-clustered combines badly with a property of the target ontology: in the target ontology, almost every primary concept can be optionally tagged with a marker class Partonomie (partonomy), indicating that the object is to be considered a container element in the building's structure. When using govterm-clustered, rules with the marker interface yield the same similarity value as rules without the marker interface. Now because we want to prefer longer matches (see section 7.1.2) when the similarity computation results in a tie, govterm-clustered always prefers the augmented version (including a Partonomie element). This is a deficit of govterm-clustered in combination with the refinement process. Depcluster similarity avoid this problem by devaluing such unjustified dangling clusters.

However, both govterm-clustered and depcluster suffer from another problem that is more strongly present in the target ontology: non-atomicity of refinement rules.

For example, the target ontology uses subclassing to represent stone types. That is there is a StoneType abstract base class and a Bucher $\sqsubseteq$ StoneType specific stone type class. Unfortunately, OWL2 does not allow the formulation of abstract classes and thus, the respective refinement is performed in two steps, since a standalone instance of StoneType is a valid model. Consequently, there are two clusters on the target side for a single cluster on the source side representing the same piece of information. In general, depcluster is designed to prefer to match the more refined cluster (e.g. Bucher$(?x)$, StoneType$(?x)$), but since there is only a single axiom representing the difference, this does not always work out.

Additionally, depcluster falls into a trap set by how a certain, well-used feature is implemented in the source and target ontologies. Some elements in the target ontology can appear either in an inventory (Bestand) or in a measurement (Massnahme). This is implemented as two marker concepts[15]. In the source ontology, the inventory taking is implicit, with only the Massnahme explicitly tagged, e.g. Vierung $-$ Massnahme. Because of this, depcluster similarity cannot discern if Vierung$(?x)$ (without Massnahme should be mapped to Vierung$(?x)$, Bestand$(?x)$ or to Vierung$(?x)$, Massnahme$(?x)$. The information is not explicitly tagged in the

[15]axiomatised as Vierung $\sqsubseteq$ Bestand $\sqcup$ Massnahme

source ontology. Which alternative is chosen is more or less random, and because the phenomenon affects the most complex primary trees found in both ontologies, its effect is visible in many refinement rules, skewing the evaluation results.

8.5 Summary

In this thesis, I have shown a new path to derive complex ontology alignments from simple correspondences. I have done so by employing and –if necessary– modifying and extending existing formal methods. Finally, I have also shown and evaluated new ways to leverage semantic information to compare description logic models, which proved useful in the derivation of new, complex mappings between ontological models.

In a first step, I have formalized a relationship between conjunctive query terms and completion graphs (chapter 5) in both directions. I have developed a method to generate completion graphs from conjunctive query terms (section 5.1.1). The other direction proved more complex and also shows the limitations of the expressive power of conjunctive queries in comparison to completion graphs (section 5.1.2). The effects of these limitations for application of the method for ontology alignment have been discussed (section 5.1.3 to section 5.1.6). For some of the limitations, potential workarounds have also been developed.

To facilitate the extraction of conjunctive query terms from saturated completion graphs, an existing DL tableau method was extended. The modified "mapping tableau" (section 5.2) extracts the necessary information to map a completion graph into (a set of) conjunctive terms. It also makes potential problems while mapping a completion graph to a conjunctive term explicit. Having access to the internal workings of the reasoning process (section 4.3.1) proved crucial to obtain the respective results.

Having developed the basic tool set for "model based" ontology alignment, the second step was the development of a model-based refinement process (section 6.1). I have shown that this newly developed process

- can integrate background information (via tableau completion),

- is extensible,
- facilitates relatively simple refinement rules (section 6.2),
- avoids inconsistent mappings, and
- can be used interactively (section 6.3) as well as in automatic mode.

The result is a new, expressive formalism to derive complex correspondences iteratively from simpler ones obtained from "traditional" ontology matchers.

Finally, the model-based refinement process has been put to the test. For evaluation, I have designed multiple completion graph-based similarity measures. These range from simple unstructured comparison of axiom sets (section 7.2.1) to iteratively more complex methods that incorporate more semantic information from both reasoning with the modified tableau (section 5.2) as well as from the refinement process (section 6.1) itself.

Empiric evaluation (chapter 8) of the full process has shown that

- model-based refinement is a workable and effective method to derive complex correspondences between ontological models,
- the above is in particular true for interactive generation of alignments. Presenting users with refinement choices is a natural and effective method of creating complex alignments, and
- incorporating semantic information in similarity measures between completion graphs is beneficial.

We have seen that evaluation of the semantic structure of a completion graph by using the governing terms of the graph has brought a big benefit in matcher quality almost across all evaluation runs (govterm-flat similarity).

Results for the even more elaborate methods govterm-clustered and depcluster are more mixed. The evaluation has shown a clear deficiency for govterm-clustered, in not discouraging rules with dangling clusters from being preferred.

While depcluster similarity remedies this particular problem, alignment quality is only improved in certain cases. However, the generated incorrect alignments provide important information. The errors made by the more complex similarity measures often showed actual modelling deficiencies (missing explicit Bestand tagging) and have also given valuable hints as too which modelling concepts are best avoided (e.g. abstract concepts).

Also on the bright side, it must be noted that the desired reference rule was always within the top three (of sometimes over a hundred) candidates returned by the similarity measure. This makes them well suited for interactive alignment, where the user can be presented with a candidate list.

8.6 Future Work

Research on a topic can usually continue indefinitely. Even now, I can think of a few parameters to tune, a few algorithmic details to change, a few external systems to integrate and –of course– a few more evaluation runs to perform. There is also the other side, where there are clearly remaining problems but not clear path to go on. Both of which will be covered in this section.

Design of Document Ontologies Remember back from section 3.3 that we have little information on the suitability of matcher technology with regard to types of documents. With regard to digital archiving, this means that we cannot currently make assertions in two regards:

- we cannot evaluate existing documents for their suitability for format conversion using ontology matching. That is, when presented with a document ontology, we cannot check the document for features that make it hard for the document to be aligned.
- we also cannot evaluate changes to a document ontology with regard to matching. This means, that even if we have a "matchable" document in the archive, there is currently no way to find out if a future, modified ontology is still "matchable" short of actually performing the alignment.

What is still missing, is thus a method to determine suitability of document ontologies for alignment.

In this thesis, a first attempt was made in the formulation of the document ontology meta-schema (Section 3.3). However, the results from the automatic evaluation part show, that this document model is not sufficiently restrictive ensure proper automatic alignment.

Syntactic restrictions in the ontology formalism (in particular the inability to express abstract classes in DLs) meant that our refinement rules were semantically non-atomic. This in turn meant that similarity measures that rely on the fact that refinement rules introduce atomic semantic features (govterm-clustered, depcluster) can yield only limited results.

One way to improve this situation is to introduce additional information into the document models. In general, a document model is only acceptable, when

- the refinement process is able to derive all relevant concrete document fragments, and
- every refinement step is semantically atomic, that is every step introduces a single semantic feature.

If this is not the case, the problem must either be fixed in the document model or we need additional or different refinement rules that support the desired properties. Verification of the property, however, remains a semi-automatic task since a proper definition of semantic property is still elusive.

Refinement Suggestion for Interactive Mapping While the interactive matcher showed itself very useful, it also has its drawbacks. It is limited by the available refinement rules, there is no way to manually introduce a particular refinement not covered by existing refinement rules.

In addition, larger refinement paths maybe troublesome when not both ontologies are well known. A possible improvement would be to allow the user to select a graph element that he/she wants a refinement suggestion for. The refinement system the automatically uses the egd path of the selected element to generate a

candidate rule and uses the refinement search process to suggest possible refinement steps for the user side of the mapping. This also enables to use the results from a selective, automated refinement process for interactive refinement.

Primary Relations The document ontology model (section 3.3) implies that there are (primary) relations between primary trees. Finding proper correspondences and "attachment locations" for such cross-tree relations has been ignored by this thesis.

Value Transfer The problem with primary relations is related also to another: value transfer, i.e. the mapping of data property values. The current system can only handle enumerated values and even those only when the enumeration is explicitly formulated in the TBox.

In the preservation case, the set of source documents is usually fixed. It is thus possible to extract data value enumerations from the documents themselves. Even more elaborate, the refinement process needs only to explore existing knowledge base fragments.

Mapping different value ranges is difficult, however. This thesis shifts this burden fully into the –user supplied– thesaurus. If one ontology contains the data value "1500" and the other the data value "16th century", user intervention is required to assert that both data values (in context) have the same meaning.

Performance and Large Scale Matching This is an important point. The known-target version of the alignment process is reasonable fast, especially when considering the performance enhancements from section 6.2.4. For creating the reference alignment, the process was run interactively and individual steps compute reasonable fast with only a few seconds of waiting time in between.

On the other hand, our reference ontology contains only 53 (56) classes and 83 (21) properties. This is far away from the SNOMED CT test cases with around $2,000$ to $3,000$ concepts, let alone the full SNOMED ontology with $316,031$ concepts. On

the other hand, SNOMED while highly detailed is also –relatively– shallow: there are only 153 properties, i.e. one per about $2,000$ classes and the representation language is $\mathcal{EL}++$.

Nonetheless, improving refinement performance remains an open topic. More information can be re-used from previous refinement steps. It might also be possible to avoid using a custom tableau and re-use an off-the-shelf and more optimized DL reasoner.

Improved Refinement Rules The current set of two refinement rules (subclass and role successor introduction) is sufficient to match built heritage ontologies. However, many ontologies will have more elaborate representation patterns. Because the refinement process is modular, adding additional patterns is possible, when the basic constraints (merge monotonicity, total order) are observed. The alignment patterns described by Scharffe [Sch09] seem like a good starting point to derive more complex refinement rules.

Improved Semantic Matching Last, but not least there is also room for improvement in one of the primary results of this thesis. While the semantic *contraction* offered by the refinement process and governing term-based semantic clustering has been shown as a step in the right direction, I consider dependency cluster matching to be more experimental. More work is required to improve similarity precision when the refinement rules are not atomic (with regard to the expression of a semantic feature).

Thus, future work is possible along two paths: it is possible to move onto a completely different approach to make more use of semantic information from the refinement process. Future work can also try to refine the dependency cluster approach, trying to obtain more stable results and improve reproducibility.

Tagging Semantic Features As noted in section 7.2.3 and discussed further in section 7.2.3, the implemented semantic refinement rules (section section 6.2.1) do not fully comply with requirement that each refinement rule introduces a single

new semantic feature. In particular for the source ontology (see section 7.2.3), multiple refinement steps are sometimes required to introduce a single semantic feature. This is undesirable.

There are two possible approaches to solve this problem. The approaches do not exclude each other. One way is to try to guess the intention of the ontology designer with regard to the atomicity of modelling features. This involves introducing smarter refinement rules that accomplish this task. Such smarter refinement rules are to the benefit of dependency cluster similarity, which has trouble handling an atomicity difference between source and target refinements.

The smart refinement approach can be combined with explicit annotation of semantic features in the source and target ontologies. For example, if FillingMaterial (FM) (table 7.4) could be marked abstract, the problem would be solved in the example case. Supplying the ontology designer with the ability to annotate elements of an ontology that need to be present together, could significantly improve refinement quality.

In addition, it is possible to argue for a *pattern-based* approach to ontology design: to introduce a new feature into an existing ontology schema, the ontology designer instantiates a pattern. The use of the pattern could be noted in the ontologies metadata (annotations). This again would provide suitable hints for refinement rules that a certain set of elements should be introduced atomically.

SECTION A.0

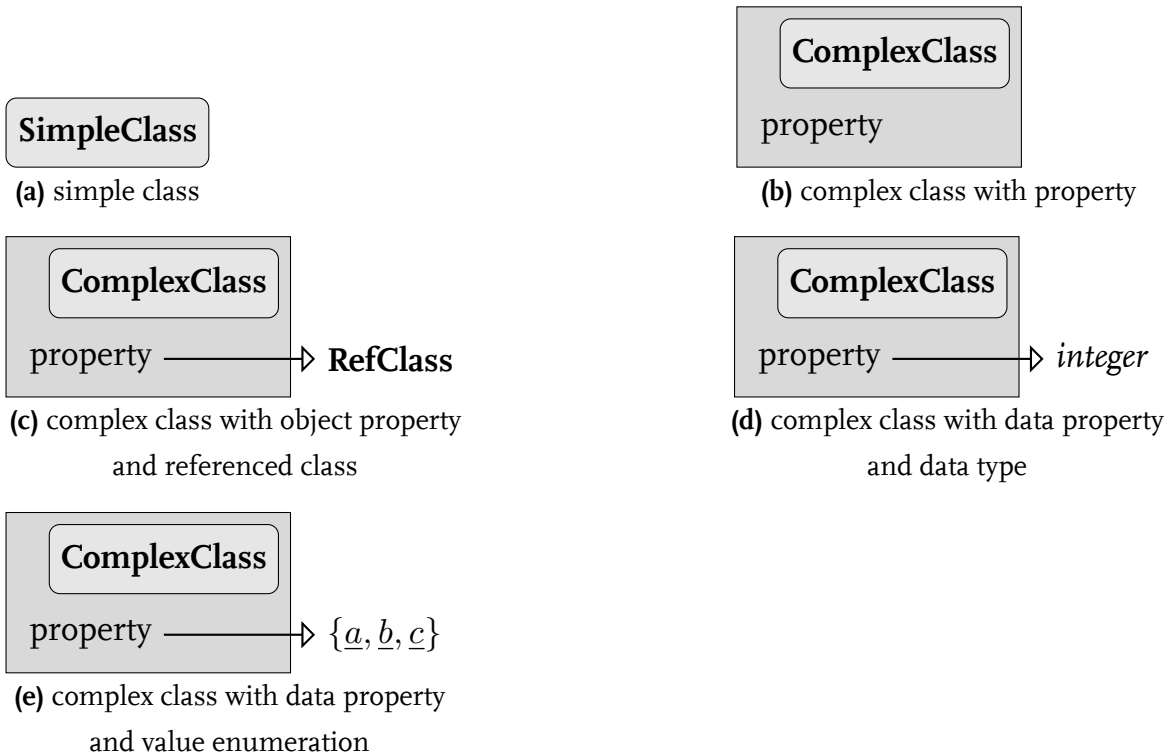
Appendix

A.1 Note on Notation

Graphical Ontology Notation

In some places in this thesis, graphical notation for ontological structures will be used. Graphical presentation is usually preferred or given in addition to formal representation. The notation somewhat borrows from UML class diagrams with ontology specific notation used where appriopriate.

Figure A.1. Graphical Notation for Ontology Elements



(a) simple class

(b) complex class with property

(c) complex class with object property and referenced class

(d) complex class with data property and data type

(e) complex class with data property and value enumeration

Relationships between classes are indicated by different types of arrows.

⟶▶	subclass relationship
⋯⋯▶	subproperty relationship
⟶▷	object property
⟶▷	data property
⟶▷ $\{\underline{a}, \underline{b}, \underline{c}\}$	data property with (enumerated) value range
⟶▷ *integer*	data property with data type

Set Arithmethic

$\boldsymbol{A = \{a_1, a_2, \ldots a_n\}}$ a set of n elements. It is true, that $a_1 \in A$, $a_2 \in A$, ..., $a_n \in A$.

$\boldsymbol{A = (a_1, a_2, \ldots a_n)}$ a tuple, i.e. an ordered set of n elements.

$\boldsymbol{\pi_i((a_1, a_2, \ldots, a_n)) = a_i}$ the projection function, π_i returns the ith item from a tuple. Given $A = (a_1, a_2, \ldots, a_n)$, we also write $A[i]$.

$\boldsymbol{\{x \mid \Phi(x)\}}$ the set of all x, for which $\Phi(x)$ holds.

$\boldsymbol{C = A \cup B}$ set union, $C \equiv_{\text{def}} \{c | c \in A \vee c \in B\}$.

$\boldsymbol{C = A \cap B}$ set intersection $C \equiv_{\text{def}} \{c | c \in A \wedge c \in B\}$.

$\boldsymbol{\emptyset}$ the empty set, also written as $\{\}$.

$\boldsymbol{|A|}$ the cardinality of set A, i.e. the number of distinguished elements $a \in A$.

Also written as $\#A$, when convenient.

$\boldsymbol{2^A = \mathfrak{P}(A)}$ The powerset of A, i.e. the set of all subsets of A, including A and the empty set $\emptyset$.

$\boldsymbol{A \times B}$ The cross product of A and B. The set of all tuples $\{(a, b) | a \in A, b \in B\}$.

$\boldsymbol{A^2 = A \times A}$ Self cross product.

$\boldsymbol{A^n = \prod_{i=1}^{n} A}$ Self set cross product, i.e. The set of all tuples $\{(a_1, \ldots, a_n) | \forall i. a_i \in A\}$.

$\boldsymbol{A^{n+} = \bigcup_{k=1}^{\infty} A^k}$ The set of all cross products of size $k = n, \ldots, \infty$. In particular, $A^+ \equiv_{\text{def}} A^{1+}$.

Multisets

$\boldsymbol{A = (S, m)}$ where S is a set of elements and $m : S \mapsto \mathbb{N}^+$ a counting function. A is a multiset. m indicates the number of occurences of each element of S in A.

$\boldsymbol{A(a) = m(a)}$ the number of occurences of the element a in the multiset $A = (S, m)$.

$\boldsymbol{a \in A}$

$$(a \in A) \Leftrightarrow A(a) > 0$$

$\boldsymbol{\mathsf{supp}(A)}$ the *support* of A, the set of elements occuring in A at least once.

$$\mathsf{supp}(A) \equiv_{\text{def}} \{a \in S | A(a) > 0\}$$

Term Manipulation

$\boldsymbol{\phi\,[\sigma]}$ A sequence of variable substitutions. $x \leftarrow \underline{a}$ indicates the variable x is to be replaced by the literal/object $\underline{a}$.

$\phi\,[\sigma]$ is the term obtained by applying all variable substitions from σ (in the prescribed order) to the term ϕ.

E.g. $\sigma = [x \leftarrow y, y \leftarrow z]$, replaces x by y, first and subsequently replaces every occurrence of y by z. $\sigma = [x \leftarrow y, y \leftarrow z]$ is therefore equivalent to $\sigma = [x \leftarrow z, y \leftarrow z]$, which is *different* from $\sigma = [y \leftarrow z, x \leftarrow y]$.

Functions

$\boldsymbol{f : A \mapsto B}$ f is a function that maps from the *domain* A onto the *range* B. Both A and B are sets.

$y = f_0 \circ ... \circ f_k(x)$ Let $f_i : X_i \mapsto Y_i$ be a set of functions and $\forall\, 0 \leq i < k \,.\, Y_i = X_{i+1}$. $y = f_0 \circ ... \circ f_k(x) : X_0 \mapsto Y_k$ is the function chain of $f_0, ..., f_k$, iff

$$y = f_0 \circ ... \circ f_k(x) \equiv y = f_k(... f_0(x))^*)$$

injection A function $f : X \mapsto Y$ is injective, iff

$$\forall\, x_0, x_1 \in X \,.\, f(x_0) = f(x_1) \Rightarrow x_0 = x_1$$

surjection A function $f : X \mapsto Y$ is surjective, iff

$$\forall\, y \in Y \,.\, \exists\, x \in X \,.\, f(x) = y$$

bijection A function is bijective, iff it is both injective and surjective.

f^{-1} Inverse function. If $f : X \mapsto Y$ is a bijective function, $f^{-1} : Y \mapsto X$ is the inverse function of f, iff

$$\forall x \in X \,.\, f \circ f^{-1}(x)$$

Because f is bijective, this is equivalent to

$$\forall y \in Y \,.\, f^{-1} \circ f(y) = y$$

Description Logics

$\underline{a}, \underline{b}, \underline{c}$ Individuals/Objects named $\underline{a}$, $\underline{b}$, and $\underline{c}$.

$A(\underline{a})$ $\underline{a}$ is a member of A, i.e. true iff the individual $\underline{a}$ belongs to the concept A. $A(x)$ is sometimes also written as $(x : A)$.

$\{\underline{a}\}$ $\{\underline{a}\}$ is the singleton class containing exactly the individual $\underline{a}$. This is called a *nominal*.

$r(\underline{a}, \underline{b})$ The role r connects the domain individual $\underline{a}$ with the range individual $\underline{b}$, i.e. true iff $(\underline{a}, \underline{b})$ is an instance of r. $r(\underline{a}, \underline{b})$ is sometimes written as $\underline{a} r \underline{b}$.

$\boldsymbol{A \sqcup B}$ concept intersection, $(\underline{a} : A \sqcup B) \Leftrightarrow (\underline{a} : A) \vee (\underline{a} : B)$

$\boldsymbol{A \sqcap B}$ concept intersection, $(\underline{a} : A \sqcap B) \Leftrightarrow (\underline{a} : A) \wedge (\underline{a} : B)$

$\boldsymbol{A \sqsubseteq B}$ subconcept assertion, $(\underline{a} : A \sqsubseteq B) \Leftrightarrow ((\underline{a} : A) \Rightarrow (\underline{a} : B))$

$\boldsymbol{\exists r \,.\, A}$ Some relationship, $(\underline{a} : \exists\, r\ .\ A) \Leftrightarrow (\exists y.r(\underline{a}, y) \wedge (y : A))$

$\boldsymbol{\forall r \,.\, A}$ Only relationship, $(\underline{a} : \forall\ r\ .\ A) \Leftrightarrow (\forall y\ .\ r(\underline{a}, y) \Rightarrow (y : A))$

$\boldsymbol{\leq_n r \,.\, A}$ Maximum cardinality constraint. The set of all concepts that have at most n distinct r-successors.

$\{x | \#\{y | r(x, y) \in E \wedge y \in A\} \leq n\}$

$\boldsymbol{\geq_n r \,.\, A}$ Minimum cardinality constraint, The set of all concepts that have at least n distinct r-successors.

$\{x | \#\{y | r(x, y) \in E \wedge y \in A\} \geq n\}$

$\top$ $true$ or the all-encompassing concept

$\bot$ $false$ or the empty concept.

A.2 Proofs

Theorem 3.1 – Knowledge Base Lattice

Given a knowledge base $KB = (\mathcal{J}, \iota_{\mathcal{C}}, \iota_{\mathcal{R}}, \iota_{\mathcal{A}})$ and a knowledge base $KB' = (\mathcal{J}', \iota'_{\mathcal{C}}, \iota'_{\mathcal{R}}, \iota'_{\mathcal{A}})$, the relation

$$(KB \leq_{\text{ext}} KB')$$
$$\Leftrightarrow$$
$$(\iota_{\mathcal{C}} \subseteq \iota'_{\mathcal{C}} \wedge \iota_{\mathcal{R}} \subseteq \iota'_{\mathcal{R}} \wedge \iota_{\mathcal{A}} \subseteq \iota'_{\mathcal{A}})$$

is a partial order.

Proof:

$KB \leq_{\text{ext}} KB$

- $KB \leq_{\text{ext}} KB$
- $\iota_{\mathcal{C}} \subseteq \iota_{\mathcal{C}}$, $\iota_{\mathcal{R}} \subseteq \iota_{\mathcal{R}}$, and $\iota_{\mathcal{A}} \subseteq \iota_{\mathcal{A}}$, (from the definition)

$KB \leq_{\text{ext}} KB' \wedge KB' \leq_{\text{ext}} KB'' \Rightarrow KB \leq_{\text{ext}} KB''$

- $KB \leq_{\text{ext}} KB' \wedge KB' \leq_{\text{ext}} KB''$
- $\iota_{\mathcal{C}} \subseteq \iota'_{\mathcal{C}}$, $\iota_{\mathcal{R}} \subseteq \iota'_{\mathcal{R}}$, and $\iota_{\mathcal{A}} \subseteq \iota'_{\mathcal{A}}$, (from the definition) as well as $\iota'_{\mathcal{C}} \subseteq \iota''_{\mathcal{C}}$, $\iota'_{\mathcal{R}} \subseteq \iota''_{\mathcal{R}}$, and $\iota'_{\mathcal{A}} \subseteq \iota''_{\mathcal{A}}$, (from the definition).
- $\iota_{\mathcal{C}} \subseteq \iota''_{\mathcal{C}}$, $\iota_{\mathcal{R}} \subseteq \iota''_{\mathcal{R}}$, and $\iota_{\mathcal{A}} \subseteq \iota''_{\mathcal{A}}$, (since $\subseteq$ is a partial order).

$KB \leq_{\text{ext}} KB' \wedge KB' \leq_{\text{ext}} KB \Rightarrow KB = KB'$

- wlg assume that all of $\iota_{\mathcal{C}}$, $\iota_{\mathcal{R}}$, and $\iota_{\mathcal{A}}$ and all of $\iota'_{\mathcal{C}}$, $\iota'_{\mathcal{R}}$, and $\iota'_{\mathcal{A}}$ are disjoint, then

  $(KB \leq_{\text{ext}} KB') \Leftrightarrow ((\iota_{\mathcal{C}} \cup \iota_{\mathcal{R}} \cup \iota_{\mathcal{A}}) \subseteq (\iota'_{\mathcal{C}} \cup \iota'_{\mathcal{R}} \cup \iota'_{\mathcal{A}}))$
- If the interpretations are disjoint and both $((\iota_{\mathcal{C}} \cup \iota_{\mathcal{R}} \cup \iota_{\mathcal{A}}) \subseteq (\iota'_{\mathcal{C}} \cup \iota'_{\mathcal{R}} \cup \iota'_{\mathcal{A}}))$ and $((\iota'_{\mathcal{C}} \cup \iota'_{\mathcal{R}} \cup \iota'_{\mathcal{A}}) \subseteq (\iota_{\mathcal{C}} \cup \iota_{\mathcal{R}} \cup \iota_{\mathcal{A}}))$ then $\iota_{\mathcal{C}} = \iota'_{\mathcal{C}}$, $\iota_{\mathcal{R}} = \iota'_{\mathcal{R}}$, and $\iota_{\mathcal{A}} = \iota'_{\mathcal{A}}$. Hence also $KB = KB'$

■

Theorem 4.1 – LillyTab Soundness, Correctness and Completeness

The rules in table 4.3 (page 115) together with table 4.4 (page 116) are a sound, complete, and terminating tableau decision procedure for $\mathcal{SHOF}$.

Proof: The proof is based on the proof for the original $\mathcal{SHOQ}$-tableau [HS05b]. The original tableau is sound and complete for $\mathcal{SHOQ}$. The original paper makes the same assumption that unfolding of TBox-terms happens automatically into $\mathcal{L}$. The original tableau is also sound and complete for general TBoxes.

LillyTab's tableau is based on an embedding of $\mathcal{SHOF}$ into $\mathcal{SHOQ}$. LillyTab's tableau is a simplification of the full $\mathcal{SHOQ}$-tableau, restricted to those rules and conditions that are applicable for $\mathcal{SHOF}$. Intuitively, LillyTab implements a variant of the $\mathcal{SHOQ}$-tableau that works only for $\mathcal{SHOF}$-concepts, but produces the same results as the full tableau, when fed with a $\mathcal{SHOF}$ knowledge base and TBox.

It is necessary to show that LillyTab performs the same operation that the $\mathcal{SHOQ}$ tableau would and comes to the same results as the full $\mathcal{SHOQ}$-tableau, when the input concepts are in $\mathcal{SHOF}$.

We have to prove that

1. For those $\mathcal{SHOQ}$-rules that are missing from LillyTab ($choose$, $\geq$), that these cannot be invoked for any $\mathcal{SHOF}$-concept.
2. If all input concepts are in $L_{\mathcal{SHOF}}$, the $\mathcal{SHOQ}$-tableau does not introduce any concepts outside of $L_{\mathcal{SHOF}}$, i.e. the $\mathcal{SHOQ}$ tableau remains in $L_{\mathcal{SHOF}}$. This includes that no inequality constraints are introduced at any point in the tableau.
3. For those rules that have been modified in LillyTab, that they behave in the same way as the unmodified $\mathcal{SHOQ}$-rules, if only $\mathcal{SHOF}$-concepts are present.

1. **Removed rules are unnecessary**

   Two rules have been omitted from LillyTab, namely $\geq$ and $choose$. For the test, assume that the completion graph does contain only $\mathcal{SHOF}$ concepts (condition (2), above):

   $\geq$ The original rule reads

   if
   1. x is not blocked
   2. $\geq_n r \,.\, C \in \mathcal{L}_k[x]$
   3. There are no n r-successors $\{y_0, \ldots, y_n\}$ of x, such that $C \in \mathcal{L}_k[y_i]$ and not $\forall\, i, j \,.\, 1 \leq i < j \leq n \Rightarrow y_i \dot{\neq} y_j$

   then create n new nodes $y_1, \ldots, y_n$ with $\mathcal{L}_k[y_i] \leftarrow \{C\}$, $E_k \leftarrow E_k \cup \{r(x, y_i)\}$, and $y_i \dot{\neq} y_j$ for all $1 \leq i < j \leq n$

The precondition of $\geq$ lists $\geq_n r\,.\,C.\ \geq_n$ is not available in $\mathcal{SHOF}$. The $\geq$ rule can thus never fire for $\mathcal{SHOF}$

If $n = 1$, condition (2) reads $\exists\, r\,.\,C$. In this case, $\geq$ falls together with the with $\exists$-rule, because no inequality constraints are generated (there is only one successor).

$\geq$ can therefore be ignored for $\mathcal{SHOF}$.

choose The original *choose*-rule reads

if
1. x is not blocked
2. $\{\geq_n r\,.\,C, \leq_n r\,.\,C\} \cap \mathcal{L}_k[x] \neq \emptyset$
3. $\exists\, y\,.\,r(x,y) \in E_k \wedge \{C, \neg C\} \cap \mathcal{L}_k[y] = \emptyset$

then Create two fresh successor graphs $G_{k+1} \leftarrow G_k$ and $G_{k+2} \leftarrow G_k$ and set
$\mathcal{L}_{k+1} \leftarrow \mathcal{L}_k \cup \{(y : C)\}$,
$\mathcal{L}_{k+2} \leftarrow \mathcal{L}_k \cup \{(y : \neg C)\}$, and
$\mathbb{G} \leftarrow (\mathbb{G} - G_k) \cup \{G_{k+1}, G_{k+2}\}$.

If $n \geq 2$, *choose* is not applicable, since qualified number restrictions with $n > 1$ are not available on $\mathcal{SHOF}$.

If $n = 1$, condition (2) can be fulfilled

- when r is a functional role. In this case, $C = \top$ ($\mathsf{functional}(r)$ is equivalent to $\top \sqsubseteq (\leq_1 r\,.\top)$). Since $\{\top, \bot\} \cap \mathcal{L}_k[y]$ is never empty and condition (3) always fails.

- when $\mathcal{L}_k[x]$ contains an existential restriction $\exists\, r\,.\,C \in \mathcal{L}_k[x]$ ($\exists\, r\,.\,C$ is equivalent to $\geq_1 r\,.\,C$)

  The intention of the *choose*-rule is it to prevent clashes that stem from inconsistent interaction of $\geq_n$ and $\leq_n$ concepts. Without *choose*, the concept description $(\geq_3 r.\ \top) \sqcap (\leq_1 r.\ C) \sqcap (\leq_1 r.\ \neg C)$ would be deemed consistent [BS01].

  However, the choose rule in the original $\mathcal{SHOQ}$ tableau is too strict.

Following later tableau implementations ([BS01, HS05b, HKS06]) it is sufficient to use $\{\leq_n r \,.\, C\} \cap \mathcal{L}_k[x] \neq \emptyset$, instead. Since we have only $(\leq_1 \,.\, \top)$ (functional(r)), introducing $\neg\top = \bot$ will always result in a clash and can therefore be avoided.

This makes the $choose$-rule unnecessary for $\mathcal{SHOF}$.

2. **Remain inside $L_{\mathcal{SHOF}}$ and no inequality constraints are introduced**.

   Only the $\geq$ rule can introduce inequality constraints. As shown in (1), $\geq$ can safely be omitted, because it requires a minimum cardinality restriction as a precondition, which is not a $\mathcal{SHOF}$ concept.

   The only concepts in $\mathcal{SHOQ}$ that are not also in $\mathcal{SHOF}$ are the cardinality restrictions $\leq, \geq$. No rule can introduce additional cardinality restrictions. Consequently, every cardinality restriction in the tableau either existed before tableau completion or was a subterm of some pre-existing concept.

   Since the initial concepts are all in $\mathcal{SHOF}$, only $\mathcal{SHOF}$ concepts can be introduced again by any tableau rule. When applying the full $\mathcal{SHOQ}$ tableau to $\mathcal{SHOF}$-concepts, all concepts encountered during tableau operation are therefore $\mathcal{SHOF}$ concepts

3. **Modified rules behave equivalently**

   $\exists$ Original rule:

   if 1. x is not blocked
   2. $\exists\, r \,.\, C \in \mathcal{L}_k[x]$
   3. $\nexists y \,.\, r(x,y) \in E_k \wedge C \in \mathcal{L}_k[y]$

   then create a new node z and set $\mathcal{L}_k[z] \leftarrow \{C\}$ and $E_k \leftarrow E_k \cup \{r(x,z)\}$

   The LillyTab tableau adds an optimization path that prevents immediate firing of the $\mathcal{F}$-rule after generating a second successor for a functional role. Instead, the concept is directly propagated to an existing successor (which would become the merge target, anyway).

   $\exists$-behaviour is equivalent to the original rule.

$\leq$ The $\leq$-rule is replaced by the $\mathcal{F}$-rule. The original rule reads

if
1. x is not blocked
2. $\leq_n r \,.\, C \in \mathcal{L}[x]$
3. x has $n+1$ r-successors $y_0, \ldots, y_n$ such that $C \in y_i$ for all y_i
4. $\exists\, i, j \,.\, \neg(y_i \dot{\neq} y_j)$
5. if only one of y_i, y_j is not anonymous, it is y_i (i.e. if only one of y_i, y_j is a nominal node, it is y_i.

then $G_k \leftarrow \mathsf{merge}(G_k, y_i, y_j)$

Since $\mathcal{SHOF}$ only has $\leq_1 r \,.\, \top$ (3) holds, whenever a functional node has more than one successor, inequality ($\dot{\neq}$) can only be introduced by $\geq_n$, which is missing from the modified tableau. (5) is handled by merge, which automatically orders its input nodes (see function merge (function 4.2)).

In $\mathcal{SHOF}$, the original $\leq$-rule is therefore equivalent to the new $\mathcal{F}$-rule.

The modified rules ($\exists$, $\leq$) show the same behaviour for $\mathcal{SHOF}$ that the original rules do for $\mathcal{SHOQ}$. The preconditions of the removed ($choose$, $\geq$) are never satisfied for $\mathcal{SHOF}$. Consequently, the tableau rules are an implementation of the $\mathcal{SHOQ}$ tableau applied to the $\mathcal{SHOF}$ language.

■

Theorem 5.1 – Completion Graph Mapping

A completion graph G_q constructed from a query Q using function generateGraph (function 5.1) has in its interpretation the same knowledge base subsets that are matched by Q.

Proof: **Base cases:** If $Q = C(\underline{a})$, then $G_q = \left(\{x\}, \emptyset, \left\{\left(x_{\{\underline{a}\}} : \{\underline{a}\}\right)\right\}\right)$ This implies $\underline{a} \in \iota(x_{\{\underline{a}\}})$.

If $Q = C(?x)$, then $G_q = \{\{x\}, \emptyset, \{(x : C)\}\}$ This implies $x \in \iota(C)$. $Q = C(?x)$ implies that $\forall\, \underline{a}\,.\, \underline{a} = ?x \implies \underline{a} \in \iota(C)$. Since $x = \sigma_q[?x] = \underline{a}$, the interpretations are equivalent.

If $Q = r(\xi_0, \xi_1)$, then $G_q = (\{x_0, x_1\}, \{r(x_0, x_1)\}, \emptyset)$ with $x_0 = \sigma_q(\xi_0)$ and $x_1 = \sigma_q(\xi_1)$. This implies that $(x_0, x_1) \in \iota(r)$.

Additionally, $Q = r(\xi_0, \xi 1)$ implies that $\forall\, \underline{a}, \underline{b}\,.\, (\xi_0 = \underline{a} \wedge \xi_1 = \underline{b}) \Rightarrow (\underline{a}, \underline{b}) \in \iota(r)$. Since $x_0 = \sigma_q[\xi_0] = \underline{a}$ and $x_1 = \sigma_q[\xi_1] = \underline{b}$ again $(x_0, x_1) \in \iota(r)$, the interpretations are equivalent.

Note that the interpretation of the union of two query terms Q_1 and Q_2 is the intersection of both interpretations. This follows directly from the definition of the conjunctive query.

The interpretation of the union of two completion graphs G_1 and G_2 is the intersection of both interpretations. This follows directly from definition 4.15.

Proving the base cases would therefore be sufficient.

Induction step: Let G_q be a completion graph transformed from a query Q.

1. Add $C(\xi)$ to Q to form Q'. This means that now $\forall\, \underline{a}\,.\, \underline{a} = \xi \Rightarrow \underline{a} \in \iota(C)$ Adding $C(\xi)$ is equivalent to adding C to $\mathcal{L}_q[\sigma_q(\xi)]$. This causes the additional constraint that $\sigma_q[\xi] \in \iota(C)$. Since $\xi = \sigma_q[\xi]$, the interpretation changes in the same way on both sides.

2. Add $r(\xi_0, \xi_1)$ to Q to form Q'. This causes the additional constraint that $\forall\, \underline{a}, \underline{b}\,.\, (\xi_0 = \underline{a} \wedge \xi_1 = \underline{b}) \Rightarrow (\underline{a}, \underline{b}) \in \iota(r)$.

   Adding $r(\xi_0, \xi_1)$ is equivalent to adding $r(\sigma_q(\xi_0), \sigma_q(\xi_1))$ to E_q. This causes the additional constraint that $\big(\sigma_q(\xi_0), \sigma_q(\xi_1)\big) \in \iota(r)$. Since $x_0 = \sigma_q(\xi_0)$ and $x_1 = \sigma_q(\xi_1)$, both interpretations change in the same way on both sides.

■

Table A.1. Evaluation Results for Flat Cosine Similarity

			equal	correct	Prec
A_1	normalizer	cosine	38	0	0.43
A_1	normalizer	cosine-tfidf	1	0	0.01
A_1	normalizer	softcosine	38	0	0.43
A_1	stemmer	cosine	38	0	0.43
A_1	stemmer	cosine-tfidf	1	0	0.01
A_1	stemmer	softcosine	38	0	0.43
A_1	stemmer-decomposer	cosine	49	1	0.56
A_1	stemmer-decomposer	cosine-tfidf	1	0	0.01
A_1	stemmer-decomposer	softcosine	49	1	0.56
A_2	normalizer	cosine	49	1	0.77
A_2	normalizer	cosine-tfidf	24	0	0.38
A_2	normalizer	softcosine	49	1	0.77
A_2	stemmer	cosine	51	1	0.80
A_2	stemmer	cosine-tfidf	24	0	0.38
A_2	stemmer	softcosine	51	1	0.80
A_2	stemmer-decomposer	cosine	54	1	0.84
A_2	stemmer-decomposer	cosine-tfidf	24	0	0.38
A_2	stemmer-decomposer	softcosine	54	1	0.84

A.3 Raw Evaluation Results

For all tables in this section, the first columns indicate the setup of matcher parameters as by table 8.1. The first numeric value is the number of generated correspondences equal to one of the reference rules. The second value is the number of generated correspondences that are correct, but where the target side is underrefined. For known-source mappings the second number is omitted, as it is always zero.

Table A.2. Evaluation Results for Governing Term Flat Cosine Similarity (part 1, A_1)

					equal	correct	Prec
A_1			normalizer	cosine	22	0	0.25
A_1			normalizer	cosine-tfidf	1	0	0.01
A_1			normalizer	softcosine	16	0	0.18
A_1			stemmer	cosine	16	0	0.18
A_1			stemmer	cosine-tfidf	1	0	0.01
A_1			stemmer	softcosine	20	0	0.23
A_1			stemmer-decomposer	cosine	20	0	0.23
A_1			stemmer-decomposer	cosine-tfidf	1	0	0.01
A_1			stemmer-decomposer	softcosine	12	0	0.14
A_1	egd		normalizer	cosine	50	0	0.57
A_1	egd		normalizer	cosine-tfidf	1	0	0.01
A_1	egd		normalizer	softcosine	50	0	0.57
A_1	egd		stemmer	cosine	50	0	0.57
A_1	egd		stemmer	cosine-tfidf	1	0	0.01
A_1	egd		stemmer	softcosine	50	0	0.57
A_1	egd		stemmer-decomposer	cosine	55	1	0.62
A_1	egd		stemmer-decomposer	cosine-tfidf	1	0	0.01
A_1	egd		stemmer-decomposer	softcosine	55	1	0.62
A_1		children	normalizer	cosine	22	0	0.25
A_1		children	normalizer	cosine-tfidf	1	0	0.01
A_1		children	normalizer	softcosine	19	0	0.22
A_1		children	stemmer	cosine	21	0	0.24
A_1		children	stemmer	cosine-tfidf	1	0	0.01
A_1		children	stemmer	softcosine	18	0	0.20
A_1		children	stemmer-decomposer	cosine	16	0	0.18
A_1		children	stemmer-decomposer	cosine-tfidf	1	0	0.01
A_1		children	stemmer-decomposer	softcosine	17	0	0.19
A_1	egd	children	normalizer	cosine	51	0	0.58
A_1	egd	children	normalizer	cosine-tfidf	1	0	0.01
A_1	egd	children	normalizer	softcosine	51	0	0.58
A_1	egd	children	stemmer	cosine	55	0	0.62
A_1	egd	children	stemmer	cosine-tfidf	1	0	0.01
A_1	egd	children	stemmer	softcosine	55	0	0.62
A_1	egd	children	stemmer-decomposer	cosine	58	1	0.66
A_1	egd	children	stemmer-decomposer	cosine-tfidf	1	0	0.01
A_1	egd	children	stemmer-decomposer	softcosine	58	1	0.66

Table A.3. Evaluation Results for Governing Term Flat Cosine Similarity (part 2, A_2)

					equal	correct	Prec
A_2			normalizer	cosine	33	13	0.52
A_2			normalizer	cosine-tfidf	24	0	0.38
A_2			normalizer	softcosine	37	9	0.58
A_2			stemmer	cosine	36	10	0.56
A_2			stemmer	cosine-tfidf	24	0	0.38
A_2			stemmer	softcosine	35	11	0.55
A_2			stemmer-decomposer	cosine	31	13	0.48
A_2			stemmer-decomposer	cosine-tfidf	24	0	0.38
A_2			stemmer-decomposer	softcosine	37	7	0.58
A_2	egd		normalizer	cosine	54	0	0.84
A_2	egd		normalizer	cosine-tfidf	24	0	0.38
A_2	egd		normalizer	softcosine	54	0	0.84
A_2	egd		stemmer	cosine	58	0	0.91
A_2	egd		stemmer	cosine-tfidf	24	0	0.38
A_2	egd		stemmer	softcosine	58	0	0.91
A_2	egd		stemmer-decomposer	cosine	60	0	0.94
A_2	egd		stemmer-decomposer	cosine-tfidf	24	0	0.38
A_2	egd		stemmer-decomposer	softcosine	60	0	0.94
A_2		children	normalizer	cosine	36	9	0.56
A_2		children	normalizer	cosine-tfidf	24	0	0.38
A_2		children	normalizer	softcosine	39	6	0.61
A_2		children	stemmer	cosine	33	13	0.52
A_2		children	stemmer	cosine-tfidf	24	0	0.38
A_2		children	stemmer	softcosine	34	12	0.53
A_2		children	stemmer-decomposer	cosine	34	12	0.53
A_2		children	stemmer-decomposer	cosine-tfidf	24	0	0.38
A_2		children	stemmer-decomposer	softcosine	35	11	0.55
A_2	egd	children	normalizer	cosine	61	1	0.95
A_2	egd	children	normalizer	cosine-tfidf	24	0	0.38
A_2	egd	children	normalizer	softcosine	61	1	0.95
A_2	egd	children	stemmer	cosine	61	1	0.95
A_2	egd	children	stemmer	cosine-tfidf	24	0	0.38
A_2	egd	children	stemmer	softcosine	61	1	0.95
A_2	egd	children	stemmer-decomposer	cosine	58	1	0.91
A_2	egd	children	stemmer-decomposer	cosine-tfidf	24	0	0.38
A_2	egd	children	stemmer-decomposer	softcosine	58	1	0.91

Table A.4. Evaluation Results for Governing Term Clustered Cosine Similarity (part 1, A_1)

					equal	correct	Prec
A_1			normalizer	cosine	21	0	0.24
A_1			normalizer	cosine-tfidf	1	0	0.01
A_1			normalizer	softcosine	11	0	0.12
A_1			stemmer	cosine	17	0	0.19
A_1			stemmer	cosine-tfidf	1	0	0.01
A_1			stemmer	softcosine	25	0	0.28
A_1			stemmer-decomposer	cosine	18	0	0.20
A_1			stemmer-decomposer	cosine-tfidf	1	0	0.01
A_1			stemmer-decomposer	softcosine	11	0	0.12
A_1	egd		normalizer	cosine	50	0	0.57
A_1	egd		normalizer	cosine-tfidf	1	0	0.01
A_1	egd		normalizer	softcosine	50	0	0.57
A_1	egd		stemmer	cosine	50	0	0.57
A_1	egd		stemmer	cosine-tfidf	1	0	0.01
A_1	egd		stemmer	softcosine	50	0	0.57
A_1	egd		stemmer-decomposer	cosine	55	1	0.62
A_1	egd		stemmer-decomposer	cosine-tfidf	1	0	0.01
A_1	egd		stemmer-decomposer	softcosine	55	1	0.62
A_1		children	normalizer	cosine	14	0	0.16
A_1		children	normalizer	cosine-tfidf	24	0	0.27
A_1		children	normalizer	softcosine	8	0	0.09
A_1		children	stemmer	cosine	20	0	0.23
A_1		children	stemmer	cosine-tfidf	16	0	0.18
A_1		children	stemmer	softcosine	6	0	0.07
A_1		children	stemmer-decomposer	cosine	16	0	0.18
A_1		children	stemmer-decomposer	cosine-tfidf	12	0	0.14
A_1		children	stemmer-decomposer	softcosine	11	0	0.12
A_1	egd	children	normalizer	cosine	72	2	0.82
A_1	egd	children	normalizer	cosine-tfidf	73	1	0.83
A_1	egd	children	normalizer	softcosine	72	2	0.82
A_1	egd	children	stemmer	cosine	71	1	0.81
A_1	egd	children	stemmer	cosine-tfidf	68	2	0.77
A_1	egd	children	stemmer	softcosine	70	2	0.80
A_1	egd	children	stemmer-decomposer	cosine	76	2	0.86
A_1	egd	children	stemmer-decomposer	cosine-tfidf	75	2	0.85
A_1	egd	children	stemmer-decomposer	softcosine	73	2	0.83

Table A.5. Evaluation Results for Governing Term Clustered Cosine Similarity (part 2, A_2)

					equal	correct	Prec
A_2			normalizer	cosine	32	14	0.50
A_2			normalizer	cosine-tfidf	24	0	0.38
A_2			normalizer	softcosine	38	8	0.59
A_2			stemmer	cosine	39	7	0.61
A_2			stemmer	cosine-tfidf	24	0	0.38
A_2			stemmer	softcosine	34	12	0.53
A_2			stemmer-decomposer	cosine	35	9	0.55
A_2			stemmer-decomposer	cosine-tfidf	24	0	0.38
A_2			stemmer-decomposer	softcosine	34	10	0.53
A_2	egd		normalizer	cosine	54	0	0.84
A_2	egd		normalizer	cosine-tfidf	24	0	0.38
A_2	egd		normalizer	softcosine	54	0	0.84
A_2	egd		stemmer	cosine	58	0	0.91
A_2	egd		stemmer	cosine-tfidf	24	0	0.38
A_2	egd		stemmer	softcosine	58	0	0.91
A_2	egd		stemmer-decomposer	cosine	60	0	0.94
A_2	egd		stemmer-decomposer	cosine-tfidf	24	0	0.38
A_2	egd		stemmer-decomposer	softcosine	60	0	0.94
A_2		children	normalizer	cosine	37	9	0.58
A_2		children	normalizer	cosine-tfidf	35	11	0.55
A_2		children	normalizer	softcosine	37	9	0.58
A_2		children	stemmer	cosine	34	12	0.53
A_2		children	stemmer	cosine-tfidf	38	8	0.59
A_2		children	stemmer	softcosine	41	5	0.64
A_2		children	stemmer-decomposer	cosine	37	9	0.58
A_2		children	stemmer-decomposer	cosine-tfidf	34	12	0.53
A_2		children	stemmer-decomposer	softcosine	37	9	0.58
A_2	egd	children	normalizer	cosine	41	20	0.64
A_2	egd	children	normalizer	cosine-tfidf	51	10	0.80
A_2	egd	children	normalizer	softcosine	48	14	0.75
A_2	egd	children	stemmer	cosine	47	13	0.73
A_2	egd	children	stemmer	cosine-tfidf	44	17	0.69
A_2	egd	children	stemmer	softcosine	38	22	0.59
A_2	egd	children	stemmer-decomposer	cosine	46	13	0.72
A_2	egd	children	stemmer-decomposer	cosine-tfidf	47	12	0.73
A_2	egd	children	stemmer-decomposer	softcosine	43	16	0.67

Table A.6. Evaluation Results for Dependency Cluster Similarity

			equal	correct	Prec
A_1	normalizer	cosine	71	1	0.81
A_1	normalizer	cosine-tfidf	79	1	0.90
A_1	normalizer	softcosine	75	1	0.85
A_1	stemmer	cosine	67	1	0.76
A_1	stemmer	cosine-tfidf	80	1	0.91
A_1	stemmer	softcosine	69	0	0.78
A_1	stemmer-decomposer	cosine	76	0	0.86
A_1	stemmer-decomposer	cosine-tfidf	81	1	0.92
A_1	stemmer-decomposer	softcosine	77	0	0.88
A_2	normalizer	cosine	59	1	0.92
A_2	normalizer	cosine-tfidf	46	6	0.72
A_2	normalizer	softcosine	59	1	0.92
A_2	stemmer	cosine	59	1	0.92
A_2	stemmer	cosine-tfidf	47	6	0.73
A_2	stemmer	softcosine	59	1	0.92
A_2	stemmer-decomposer	cosine	58	1	0.91
A_2	stemmer-decomposer	cosine-tfidf	48	6	0.75
A_2	stemmer-decomposer	softcosine	58	1	0.91

Table A.7. Evaluation Results With Known Source

			equal	correct	Prec
flat	A_1	cosine	55	1	0.62
flat	A_1	softcosine	55	1	0.62
govterm-flat	A_1	cosine	73	0	0.83
govterm-flat	A_1	cosine-tfidf	48	8	0.55
govterm-flat	A_1	softcosine	73	0	0.83
govterm-clustered	A_1	cosine	3	0	0.03
govterm-clustered	A_1	softcosine	5	0	0.06
depcluster	A_1	cosine	51	1	0.58
depcluster	A_1	softcosine	51	1	0.58
flat	A_2	cosine	56	0	0.88
flat	A_2	softcosine	56	0	0.88
govterm-flat	A_2	cosine	55	0	0.86
govterm-flat	A_2	cosine-tfidf	52	4	0.81
govterm-flat	A_2	softcosine	55	0	0.86
govterm-clustered	A_2	cosine	17	0	0.27
govterm-clustered	A_2	softcosine	16	0	0.25
depcluster	A_2	cosine	53	1	0.83
depcluster	A_2	softcosine	53	1	0.83

List of Definitions

2.1. Semantic Feature 38

3.1. Ontology Elements 45
3.2. Knowledge Base 48
3.3. Knowledge Base Extension 48
3.4. Ontology Frame 49
3.5. Sub- and Superclass Set 51
3.6. Sub- and Superproperty Set 51
3.7. Domain and Value Range 51
3.8. Role 52
3.9. Domain and Range 52
3.10. Language Signature 54
3.11. Ontology Constraint System 55
3.12. Consistent Knowledge Base for an Ontology Constraint System . . 55
3.13. Ontology 56
3.14. Consistent Axiom 57
3.15. Inferable Axiom 58
3.16. Alignment Correspondence 60
3.17. Alignment 60
3.18. Simple Alignment 61
3.19. Bridge Rule 62
3.20. Conservative Bridge Rule 62
3.21. Alignment Precision 65
3.22. Alignment Recall 65
3.23. Alignment f-score 65

3.24. α-Consequence 66
3.25. Ideal Alignment Precision 66
3.26. Ideal Alignment Recall 67
3.27. Inconsistent Correspondence 71
3.28. Incoherent Alignment 72

4.1. Formal Semantics for $\mathcal{ALC}$ 87
4.2. Negation Normal Form 88
4.3. Description Logic ABox 94
4.4. Description Logic TBox 94
4.5. Description Logic RBox 94
4.6. DL Sub- and Superrole Closure 95
4.7. DL Completion Graph 97
4.8. Consistency of a Completion Graph 97
4.9. Concept Inferability in a Completion Graph 98
4.10. Capacity of a Completion Graph 98
4.11. Completion Graph Extension 99
4.12. Anonymous Node in a Completion Graph 101
4.13. Concept Use 107
4.14. Definitorial Concept Inclusion 107
4.15. Formal Semantics for $\mathcal{SHOF}(\mathbf{D})$ 112
4.16. Priority of Rule Application 116
4.17. Consistency Checks for $\mathcal{SHOF}(D)$ 117
4.18. Saturated Completion Graph 118
4.19. Subset Blocking 119

5.1. Conjunctive Query 123
5.2. Query Node Mapping 124
5.3. Augmented DL Completion Graph 141
5.4. Generating Existential Dependency 146
5.5. Tuple Generating Dependency 156
5.6. Mapped Node 156
5.7. Deductive Conservative Extension 161

6.1. Refinement Graph 168
6.2. Merge Monotonic 181

7.1. Extracted Phrase Set 199
7.2. Dependency Path 204
7.3. Existential Generating Dependency 205
7.4. Term Frequency–Inverse Document Frequency 209
7.5. Soft Cosine 210

List of Theorems

3.1. Knowledge Base Lattice 47

4.1. Soundness, Completeness, and Termination of the LillyTab Tableau for $\mathcal{SHOF}(\mathbf{D})$ 114

5.1. Correctness of Completion Graph Generation from Conjunctive Query 124

6.1. Completion Graph Refinement 173

List of Figures

1.1. Part of a Built Heritage Digital Map for St. Stephan's Cathedral, Vienna 4

1.2. Structure of an Archived Digital Map in Built Heritage 6

1.3. Source Ontology Fragment – Measure — $\mathsf{GroutFilling}_s$ 11

1.4. Target Ontology Fragment – $\mathsf{GroutFilling}_t$ 12

2.1. OAIS Archive Management Overview 27

2.2. Module Dependencies 33

2.3. Interpretation with the Universal Virtual Computer 34

2.4. Distorted Graphics after Import into Different Presentation Software 36

3.1. Alignment Generation 60

3.2. Ontology Matching System Classification (figure from [Euz07]) . . 64

3.3. Structure of a Document Ontology 76

4.1. Syntax of Frame Logic 84

4.2. Syntax of the Description Logic $\mathcal{AL}$ 85

4.3. Syntax of the Description Logic $\mathcal{ALC}$ 89

4.4. Example Tableau Completion 100

4.5. Blocking in Expressive DLs 104

4.6. Tableau Thrashing without Dependency Directed Backtracking . . 110

4.7. Dependency directed backtracking 111

5.1. Alternative Representations for Completion Graphs 126

5.2. $\sqcap$-Propagation for Tableau Completion 127

5.3. $\forall$-Handling in Conjunctive Query Generation and Mismatching Knowledge Base 130

5.4. Blocked Completion Graphs and Corresponding Knowledge Bases 134

5.5. Non-determinism during Tableau Expansion with Concept Subsumption 135

5.6. Non-determinism during Tableau Expansion with Role Inheritance 137

5.7. $KB_{2^{\sqsubseteq\exists}}$: Query Coverage over Knowledge Bases 138

5.8. Query Extraction using the Mapping Tableau 148

5.9. Example Mapping as Model Graphs 159

5.10. Knowledge Bases as Source and Target in Mapping 159

6.1. Example Refinement Graph Fragment 170

6.2. Refinement Graph 174

6.3. Manual Mapping Tool: Mapping for Grouting 177

6.4. Ordering of Refinement Rules 180

6.5. Interactive Mapping Tool, generating mapping for Vierung 186

6.6. Interactive Mapping Tool, Refinement suggestions for Vierung . . . 186

6.7. Interactive Mapping Tool, Possible complex correspondence for Vierung with material Coburger 186

7.1. Examples of Existential Generating Dependency Paths 202

7.2. Generating Refinement Rules 214

7.3. Dependency Cluster Constraint Matrix 215

8.1. Effects of Extractor Parameters 228

8.2. Effects of Term Set Measure 230

8.3. Effects of Dependent Term Collection 230

8.4. Best Results by Completion Graph Similarity 232

A.1. Graphical Notation for Ontology Elements 243

List of Tables

1.1. Simple Correspondences for the Example Ontologies 7

1.2. Abbreviations for Concepts and Roles in the Initial Example 8

3.1. Simple Matching Results for the "Dom Bamberg" Ontology and Its Evolved Successor 64

3.2. Number of Reference Alignments for the OAEI "conference" Dataset. 68

4.1. Interpretation-Preservation Transformations in $\mathcal{ALC}$ 88

4.2. Tableau Rules for Absorbed Range and Domains 106

4.3. Tableau Rules for the LillyTab Reasoner, part 1 115

4.4. Tableau Rules for the LillyTab Reasoner, part 2 116

5.1. Tableau Rules for the Mapping Tableau, part 1 144

5.2. Tableau Rules for the Mapping Tableau, part 2 145

7.1. Simple Matching Results for the "Dom Bamberg" Ontology and its Evolved Successor 190

7.2. Alignment Length 192

7.3. Variations of Governing Term Cosine Similarity 211

7.4. Example Axiom-Clustered Alignment 212

7.5. Example Expanded Axiom-Clustered Alignment 213

7.6. Example Axiom Cluster Similarity Matrix 217

8.1. Matcher Parameters for the Evaluation 226

A.1. Evaluation Results for Flat Cosine Similarity 254

A.2. Evaluation Results for Governing Term Flat Cosine Similarity (part 1, A_1) 255

A.3. Evaluation Results for Governing Term Flat Cosine Similarity (part 2, A_2) 256

A.4. Evaluation Results for Governing Term Clustered Cosine Similarity (part 1, A_1) 257

A.5. Evaluation Results for Governing Term Clustered Cosine Similarity (part 2, A_2) 258

A.6. Evaluation Results for Dependency Cluster Similarity 259

A.7. Evaluation Results With Known Source 260

List of Algorithms

4.1. Procedure unfold(G, TBox) 108

4.2. Function merge(G, t, s) 113

5.1. Procedure generateGraph(Q) 125

5.2. Function augmerge(G, t, s) 142

5.3. Function mergeMap(G, x) 150

5.4. Function repr(G, x, σ_q) 151

5.5. Function generateCoreQuery(G_0, G_s, σ_q) 153

5.6. Function generateQuery(G_0, G_s) 154

5.7. Function primitiveApplyTGD((ϕ, ψ), O_s, O_t) 158

6.1. Function subClassRefinement(G, TBox, x) 178

6.2. Function roleSuccRefinement(G, TBox, x) 179

7.1. Function extractPhrase(x, $\mathbb{O}$) 199

7.2. Function extractToken(ax, $\mathbb{O}$) 200

7.3. Function analyseToken(tk, thes, decompose, lower, Stop, normalize, stem) 201

7.4. Procedure collectEGDs(G, ax, x_0) 206

Index

Symbols
KB, **48**
O, **56**
OCS, **55**
O_s, 166
O_t, 166
Corp, **207**
Dep, **141**
Mer, **141**
P, **167**
Σ_L, **54**
$\Sigma_{L^+_{\text{conjunctive}}}$, **167**
$\Sigma_{L_{\text{conjunctive}}}$, **167**
Tk(t), **207**
α-consequence, **66**
β, 61, **62**
ref$_s$, **167**
ref$_t$, **167**
sim$_{\text{depcluster}}$, **217**
sim$_G$, **169**
sim$_S$, **167**
sim$_{\text{cos,tfidf}}$, **209**
sim$_{\text{cos}}$, **208**
$\iota_{\mathcal{C}}$, **48**
$\iota_{\mathcal{R}}$, **48**
$\iota_{\mathcal{D}}$, **117**
κ, **141**, 202
$\leq_{\mathcal{C}}$, **49**, **50**
$\leq_{\mathcal{R}}$, **49**
$\leq_{\text{ext}}$, **48**
$\mathbb{A}$, 166
$(\mathbf{D})$, **93**
$\mathcal{ALC}$, **87**
$\mathcal{AL}$, **85**
$\mathcal{A}$, **45**, **48**, **55**
$\mathcal{C}$, **45**, 48, **55**
$\mathcal{EL}$, **85**
$\mathcal{E}$, **45**, **91**
$\mathcal{FL}$, **83**
$\mathcal{FL}^-$, **84**
$\mathcal{F}$, **91**
$\mathcal{H}$, **91**
$\mathcal{I}$, **45**, **48**, **55**, **90**
$\mathcal{L}$, **97**, 141
$\mathcal{N}$, **92**
$\mathcal{OF}$, **49**
$\mathcal{O}$, **92**
$\mathcal{Q}$, **92**
$\mathcal{R}$, **45**, 48, **55**, **91**
$\mathcal{SHOF}(\mathbf{D})$, **112**, 113

$\mathcal{SHOIQ}$, 92
$\mathcal{SROIQ}$, 92
$\mathcal{S}$, **90**
$\mathcal{T}$, **45**, **48**, **55**
$\mathcal{U}$, **90**
$\models$, **58**
$\overrightarrow{\mathsf{Tk}}$, **208**
ϕ, 57, 58, 61, **62**
ψ, 61, **62**
σ, **49**
σ_q^{-1}, **124**
σ_q, **124**
$C{\downarrow}$, **51**
$r{\downarrow}$, **95**
$C{\uparrow}$, **51**
$r{\uparrow}$, **95**
f-score, **65**

A

ABox, **53**
absorption, **106**
affordance, **36**
alignment, **60**
  f-score, **65**
  coherence, 70, **72**
  complex-, **61**, 74
  consistency, 70, **71**
  correspondence, **60**
  explaining, **74**
  length, **192**
  level 0, **59**
  level 1, **61**
  level 2, **61**
  parameter tuning, **73**
  precision, **65**
    ideal, **66**
  quality, **65**
  recall, **65**
    ideal, **67**
  reference-, **225**
    reduced-, **225**
  refinement, **164**
  rule
    correct-, 224
  semantic-, **70**
  simple, **61**
  user involvement, **73**
alignment rule
  length, **193**
analyser
  token-, **201**
analysis
  lexical, **198**
archive
  access, **28**
  administration, **28**
  data management, **28**
  digital-, **26**
  dissemination, **28**, **36**
  ingest, **27**
  storage, **28**
assertion
  concept-, **53**
  property-, **53**
axiom
  consistent, **57**

definitorial, **107**
filter, **199**
filtering, **200**
κ-based, **202**
inferable, **58**
justification, **214**
axiom cluster, **212**

B
backtracking
dependency directed-, **109**
bitstream
preservation, **29**
blocking, **102**, 119
branching, **100**
semantic-, **109**

C
capacity
completion graph, **98**
checksumming, **29**
Choco, **219**
class, 45
classification, **81**
cluster
axiom-, **212**
coherence, **81**
combination
semantic-, **57**
completion graph, **97**, 121
augmented, **141**
capacity, **98**
comparing, **189**
consistency, **97**
inferability, **98**
refinement, **173**
relation to queries, **121**
saturated, **118**
similarity, **190**, 207
compound word
decomposition, **198**
concept, **44**
assertion, **53**
definitorial, **107**
conjunctive query, **123**
consistency, **81**
completion graph, **97**
tableau, **117**
consistent, 57
corpus, **208**
correspondence
alignment-, **60**
length, **193**
cosine, **208**
flat-, **208**
soft-, **210**

D
data range, 117
decomposition, **198**
semantic-, **36**
depcluster, **213**, **220**
dependency
tuple generating-, **155**
dependency directed backtracking, **109**
dependency path, **204**
description logic, **83**

expressive, **90**
direction
matching-, **171**, 234
dissemination, 28
distance
edit-, **210**
document
affordance, **36**
complex digital, **4**
complex digital-, **35**
intelligibility, **21**
domain, **51**, 105
concrete-, **111**
duplication, 29

E

E, **97**, 141
edit distance, **210**
egd, **205**
element, 45
justified, **193**
primary, **75**
secondary, **75**
emulation, 32
evaluation
design, **223**
reference alignment, **225**
results, **227**
existential generating dependency, **205**
extension, **45**, 48
extraction
granularity, **195**
parameters-, 228
phrase set, **199**
preprocessing, **196**
query-, 143
resource, **195**
special preprocessing, **196**
token, **194**, **199**, **200**
token set, **199**

F

feature
semantic, 166
significant, **38**
mapping, **38**
filter
axiom-, **199**
first order logic, **7**
FOL, **7**, 82
format ageing, 1, **25**

G

GCI, 106
general concept inclusion, **106**
govterm, 131, **141**
govterm-cosine, **211**

H

heterogeneity
semantic, **20**

I

ILP, **164**
inferability
completion graph, **98**
inferable, 58
inference, **80**

information
representation, **19**
semantic-, **211**, 233
transfer, **192**
ingest, 27
instance, **44**, 45
intelligibility, **21**
contextual, **30**
dependency, **22**
module, **22**
monitoring, **31**
intelligible
management, **29**
interpretation
automated, 22
dependency, **22**
module, **22**

J

justification, 193, **214**

K

KL-ONE, **83**
knowledge base, **48**
consistent, **55**
description logic, **93**
lattice, **47**
knowledge representation, **79**
known source, **171**
known target, **172**

L

L, **54**
label
extraction, **199**
language
signature, **54**
length
alignment-, **192**
lexical
processing, 233
lexical analyser, **198**
LillyTab, **110**
tableau, **113**
literal, **45**, **55**
logic
description-, **83**
first order-, **7**, 82
FOL, 7
knowledge representation, **79**
NNF, **88**
Lucene, **198**

M

mapping
query node-, **124**
matcher
parameters, **226**
matcher-mode, **171**
known source, **171**
known target, **172**
matching
direction, **171**, 234
ontology-, **63**
pipeline, 233
semantic-, **70**, 241
merge, **113**

augmented, **142**
monotonic, **181**
merge map, 141
merge monotonic, **181**
merging
modulo-, **181**
migration, 34
model
bridge-, 38
construction, **95**

N

N, **97**
NNF, **88**
node
anonymous, 101
augmented merge, **142**
merge, **113**
normalizer, **198**

O

object property, **52**
ontology, 42, 49, **56**
ABox, **53**
alignment, **60**
concept, **44**
constraint system, **55**
document-, **74**
element, **45**
instance, **44**
knowledge base, **48**
extension, 48
lattice, **47**
literal, **45**
matching, **63**
large-scale, **68**
scalability, **68**
semantic, **69**
system classification, **63**
relation, **44**
role, **52**
TBox, **53**
view
frame-based, **47**
linked-data, **46**
logics-based, **52**
views, **45**
optimization, **119**
reasoner-, **105**
OWL
2, **92**
Lite-, **92**

P

path
dependency-, **204**
egd-, **205**
performance
refinement, 240
refinement-, **178**
phrase set, **199**
pipeline
matching, 233
precision, **65**
preservation
bitstream, **29**
checksumming, **29**

duplication, **29**
emulation, **32**
migration, **34**
planning, **28**
scrubbing, **30**
processing
lexical, 233
properties
significant, **35**
property
assertion, **53**
domain, **51**, **52**
object-, **52**
range, **52**
significant
abstract, 37
concrete, 37
source-to-target, **159**
value range, **51**

Q

query, 121
complexity, **155**
conjunctive, **123**
extraction, **124**, 143
∃, **134**
∀, **130**
¬, **129**
TBox, **132**
relation to completion graphs, **121**
query node mapping, **124**

R

range, 105
RBox, **94**
description logic, **94**
reasoning, 9, 41, **80**
Abox, 102
absorption, **106**
classification, **81**
coherence, **81**
completion graph, **97**
consistency, **81**
dependency directed backtracking, **109**
implementation, 119
lazy unfolding, **106**
LillyTab, **110**, 119
model construction, **95**
optimization, **105**, **119**
resolution, **95**
satisfiability, **80**
semantic branching, **109**
subsumption, **81**
tableau, **96**
TBox, 102
undecidability, **82**
recall, **65**
ideal, **66**, **67**
refinement, 8, 164
completion graph-, **173**
graph, **168**
matcher-mode, **171**
model-based-, **168**
over-refined-, **194**
parameters, **226**
performance, **178**, 240
process, **166**

role successor-, **177**
rule, 241
rule order, **180**
semantic, **175**, 175
semantic feature, 166
strategy, **169**, **171**
subclass introduction-, **176**
refinement rule, **172**
applicable, **181**
evidence, **181**
semantic, **175**, 175
relation, **44**, 45
primary-, 240
resolution, **95**
restriction
self-, **93**
role, **52**
subrole closure, **95**
successor refinement, **177**
superrole closure, **95**
rule
$\exists$, **115**, 145
$\forall$, **115**, 144
$\forall^+$, **115**, 144
$\mathcal{F}$, **116**, 145
$\sqcap$, **115**, 144
$\sqcup$, **115**, 144
o, **116**, 145
applicable, **181**
bridge-, **62**
conservative-, **62**
comparing, **221**
evidence, **181**
generating-, **114**
non-generating-, **114**
order, 180
refinement-, **172**, 241
semantic, **175**, 175
tableau-, **113, 114**, **144**
rule-
alignment-
correct, **222**
correct, **222**

S

satisfiability, **80**
saturation, **99**, 118
scrubbing, **30**
semantic
heterogeneity, **20**
matching, 241
semantic feature, 166
semantics
$\mathcal{ALC}$, **87**
$\mathcal{SHOF}(\mathbf{D})$, **112**
set
phrase, **199**
subclass, **51**
subproperty, **51**
subrole, **51**
superclass, **51**
superproperty, **51**
superrole, **51**
token, **199**
signature, **54**
similarity

axiom set-, **208**, 229
baseline-, **207**
clustered cosine-, **211**
completion graph, 207
completion graph-, **190**
corpus, **208**
cosine, **208**
tfidf, **208**
depcluster-, **213**, **220**
dependency cluster-, **213**
element level, **190**
govterm-cosine-, **211**
soft cosine-, **210**
tfidf, **208**
token-, **209**
soft cosine, **210**
source-to-target property, **159**
stemmer, **198**
stop word, **198**
strategy
refinement-, **171**
subclass
refinement, **176**
subsumption, **81**

T

tableau, **96**, 113
augmented-, **144**
blocking, **102**, 119
consistency, **117**
data structure, **117**
mapping-, **139**
rule, **144**
saturation, **99**
TBox, 53, **93**
description logic, **93**
term
cluster, **211**
governing, 131, **141**
tfidf, **209**
tgd, **155**, 163
token
analyser, **201**
extraction, **194**, **200**
granularity, **195**
preprocessing, **196**
resource, **195**
special preprocessing, **196**
similarity, **209**
token set, **199**
tree
primary, **76**
tuple generating dependency, **155**

U

unfolding
domain-, **105**
lazy-, **106**, 108
range-, **105**

V

V, **97**, 141
value
range, **51**
transfer, **240**
view
frame-based, **47**

linked-data, **46**
logics-based, **52**

Bibliography

[Abr05] S. L. Abrams. Establishing a Global Digital Format Registry. *Library Trends*, 54(1):125–143, 2005.

[ADMR05] D. Aumueller, H.H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908, 2005.

[AGE+12] J. L. Aguirre, B. C. Grau, K. Eckert, J. Euzenat, A. Ferrara, R. W. van Hague, L. Hollink, E. Jimenez-Ruiz, C. Meilicke, A. Nikolov, et al. Results of the Ontology Alignment Evaluation Initiative 2012. In *Proc. 7th ISWC workshop on ontology matching (OM)*, pages 73–115, 2012.

[Baa03a] F. Baader. Terminological cycles in a description logic with existential restrictions. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 325–330. Lawrence Erlbaum Associates Ltd, 2003.

[Baa03b] F. Baader, editor. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[Bar77] J. Barwise. *Handbook of Mathematical Logic*, chapter An introduction to first-order logic, pages 5–46. North Holland Publishing Company, 1977.

[Bes72] J.G.P. Best. *Some Preliminary Remarks on the Decipherment of Linear A*. Hakkert, 1972.

[BHN+94] F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, and E. Franconi. An empirical analysis of optimization techniques for terminological representation systems. *Applied Intelligence*, 4(2):109–132, 1994.

[BHP08] F. Baader, J. Hladik, and R. Penaloza. Automata can show PSPACE results for description logics. *Information and Computation*, 206(9):1045–1056, 2008.

[BL84] R.J. Brachman and H.J. Levesque. The Tractability of Subsumption in Frame-Based Description Languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI-84)*, pages 34–37, 1984.

[BS85] R.J. Brachman and J.G. Schmolze. An Overview of the KL-ONE Knowledge Representation System*. *Cognitive science*, 9(2):171–216, 1985.

[BS01] F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69(1):5–40, 2001.

[BS03] F. Baader and U. Sattler. Description logics with aggregates and concrete domains. *Information Systems*, 28(8):979–1004, 2003.

[BSW02] F. Baader, H. Sturm, and F. Wolter. Fusions of description logics and abstract description systems. *Journal of Artificial Intelligence Research*, 16:200–258, 2002.

[BW97] K. Berry and O. Weber. *Kpathsea library*, 1997.

[CAS09] I.F. Cruz, F.P. Antonelli, and C. Stroe. AgreementMaker: Efficient Matching for Large Real-World Schemas and Ontologies. *Proceedings of the VLDB Endowment*, 2(2):1586–1589, 2009.

[CDGL+13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. *Artificial Intelligence*, 195:335–360, 2013.

[CGT89] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166, 1989.

[Cim06] P. Cimiano. *Ontology Learning and Population from Text Algorithms, Evaluation, and Applications*. Springer, 2006.

[CM77] A. K. K Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.

[Con12] Consultative Committee for Space Data Systems. *Reference Model for an Open Archival Information System (OAIS)*, 2012.

[CST00] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[dBFK+05] J. de Bruijn, D. Fensel, U. Keller, M. Kifer, H. Lausen, A. Polleres, and L. Predoiu. Web Service Modeling Language (WSML). Technical report, W3C Member Submission, 2005.

[DE08] J. David and J. Euzenat. On Fixing Semantic Alignment Evaluation Measures. In *The 7th International Semantic Web Conference*, page 25, 2008.

[Doa02] A. Doan. *Learning to Map between Structured Representations of Data*. PhD thesis, University of Washington, 2002.

[DR02] H.-H. Do and E. Rahm. Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 610–621. VLDB Endowment, 2002.

[DS05] M. Dürst and M. Suignard. Internationalized resource identifiers (IRIs). Technical report, RFC 3987, January, 2005.

[EE05] M. Ehrig and J. Euzenat. Relaxed precision and recall for ontology matching. In *Proc. K-Cap 2005 workshop on Integrating ontologies, Banff (CA)*, pages 25–32, 2005.

[EMS+11] J. Euzenat, C. Meilicke, H. Stuckenschmidt, P. Shvaiko, and C. Trojahn. Ontology Alignment Evaluation Initiative: six years of experience. *Journal on Data Semantics*, 2011.

[Euz04] J. Euzenat. An API for ontology alignment. *Proc. 3rd international semantic web conference, Hiroshima (JP)*, pages 698–712, 2004.

[Euz07] J. Euzenat. Semantic precision and recall for ontology alignment evaluation. In *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 348–353, 2007.

[EV03] J. Euzenat and P. Valtchev. An integrative proximity measure for ontology alignment. *Proc. ISWC-2003 workshop on semantic information integration, Sanibel Island (FL US)*, pages 33–38, 2003.

[Fag09] R. Fagin. Tuple-Generating Dependencies. In *Encyclopedia of Database Systems*, pages 3201–3202. Springer, 2009.

[FMK+08] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou. Ontology change: classification and survey. *The Knowledge Engineering Review*, 23(02):117–152, 2008.

[fS03] International Organization for Standardization. *Reference Model for an Open Archival Information System (OAIS)*. International Organization for Standardization, 2003.

[GHKS08] B.C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence Research (JAIR)*, To Appear, 2008.

[GLS01] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, May 2001.

[GNM09] A. Ghazvinian, N.F. Noy, and M.A. Musen. Creating mappings for ontologies in biomedicine: Simple methods work. In *AMIA Annual Symposium Proceedings*, volume 2009, page 198. American Medical Informatics Association, 2009.

[GR02] F. Goasdoué and M.-C. Rousset. Compilation and approximation of conjunctive queries by concept descriptions. In *ECAI*, pages 267–271, 2002.

[Gru08] TR Gruber. Ontology. In the Encyclopedia of Database Systems, Ling Liu and M. Tamer Özsu, 2008.

[GS08] A. Gal and P. Shvaiko. Advances in Ontology Matching. *Advances in Web Semantics I*, pages 176–198, 2008.

[HA07] G. Hodge and N. Anderson. Formats for digital preservation: A review of alternatives and issues. *Information Services and Use*, 27(1):45–63, 2007.

[HGKR12] M. Hartung, A. Gross, T. Kirsten, and E. Rahm. Effective Mapping Composition for Biomedical Ontologies. In *Semantic Interoperability in Medical Informatics (SIMI-12)*, 2012.

[HJK+07] P. Hitzler, Euzenat J., M. Krötzsch, L. Serafini, H. Stuckenschmidt, H. Wache, and A. Zimmermann. Deliverable D2.2.5 Integrated View and Comparison of Alignment Semantics. Technical report, NoE Knowledge Web project, 2007.

[HKS06] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67, 2006.

[HL02] M. Hedstrom and C.A. Lee. Significant Properties of Digital Objects: Definitions, Applications, Implications. In *Proceedings of the DLM-Forum*, pages 6–8, 2002.

[HM00] V. Haarslev and R. Moller. Expressive ABox Reasoning With Number Restrictions, Role Hierarchies, and Transitively Closed Roles. In *Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 273–284. Morgan Kaufmann Publishers; 1998, 2000.

[HM01] V. Haarslev and R. Moller. RACER system description. *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, 2083:701–705, 2001.

[HMH01] M.A. Hernández, R.J. Miller, and L.M. Haas. Clio: A semi-automatic tool for schema mapping. In *ACM SIGMOD Record*, volume 30, page 607. ACM, 2001.

[Hor97] I.R. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, the University of Manchester, 1997.

[HS01] I. Horrocks and U. Sattler. Ontology reasoning in the SHOQ (D) description logic. In *International joint conference on artificial intelligence*, volume 17, pages 199–204. LAWRENCE ERLBAUM ASSOCIATES LTD, 2001.

[HS05a] P. Haase and L. Stojanovic. Consistent Evolution of OWL Ontologies. *The Semantic Web: Research and Applications: Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29-June 1, 2005: Proceedings*, 2005.

[HS05b] I. Horrocks and U. Sattler. A tableau decision procedure for SHOIQ (D). *Intl. Joint Conference on Artificial Intelligence*, 2005.

[HSG04] U. Hustadt, RA Schmidt, and L. Georgieva. A Survey of Decidable First-Order Fragments and Description Logics. *Journal of Relational Methods in Computer Science*, 1:251–276, 2004.

[HST99] I. Horrocks, U. Sattler, and S. Tobies. A Description Logic with Transitive and Converse Roles, Role Hierarchies and Qualifying Number Restrictions. Technical Report LTCS-Report 99-08, RWTH Aachen, 1999.

[HST00a] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of IGPL*, 8(3):239, 2000.

[HST00b] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In *Automated Deduction-CADE-17*, pages 482–496. Springer, 2000.

[HT00] I. Horrocks and S. Tobies. Reasoning with Axioms: Theory and Practice. *CoRR*, cs.LO/0005012, 2000.

[HV05] P. Hitzler and D. Vrandečić. Resolution-based approximate reasoning for OWL DL. In *The Semantic Web–ISWC 2005*, pages 383–397. Springer, 2005.

[JHCQ05] N. Jian, W. Hu, G. Cheng, and Y. Qu. Falcon-AO: Aligning Ontologies with Falcon. In *Integrating Ontologies Workshop Proceedings*, page 85. Citeseer, 2005.

[JHS+10] P. Jain, P. Hitzler, A. Sheth, K. Verma, and P. Yeh. Ontology alignment for linked open data. *The Semantic Web–ISWC 2010*, pages 402–417, 2010.

[JMSK09] Y.R. Jean-Mary, E.P. Shironoshita, and M.R. Kabuka. Ontology matching with semantic verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):235–251, 2009.

[JRG11] E. Jiménez-Ruiz and B. Grau. Logmap: Logic-based and scalable ontology matching. In *The Semantic Web–ISWC 2011*, pages 273–288. Springer, 2011.

[JRGZH12] E. Jiménez-Ruiz, B. Grau, Y. Zhou, and I. Horrocks. Large-scale interactive ontology matching: Algorithms and implementation. In *Proc. of ECAI*, 2012.

[JRL08] N. Jussien, G. Rochart, and X. Lorca. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, 2008.

[JS91] S. Jajodia and R. Sandhu. A Novel Decomposition of Multilevel Relations Into Single-Level Relations. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 300–313, 1991.

[JW09] K. Janowicz and M. Wilkes. SIM-DL_A: A Novel Semantic Similarity Measure for Description Logics Reducing Inter-Concept to Inter-Instance Similarity. In *The 6th Annual European Semantic Web Conference (ESWC2009)*, pages 353–367, 2009.

[Ken89] W. Kent. The Many Forms of a Single Fact. In *IEEE COMPCON*, 1989.

[Kif05] M. Kifer. Rules and Ontologies in F-Logic. *Lecture notes in computer science*, 3564:22–34, 2005.

[KKS11] Y. Kazakov, M. Krötzsch, and F. Simančík. Concurrent Classification of $\mathcal{EL}$-Ontologies. In *The Semantic Web–ISWC 2011*, pages 305–320. Springer, 2011.

[KL89] M. Kifer and G. Lausen. F-Logic: a Higher-Order Language for Reasoning About Objects, Inheritance, and Scheme. *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 134–146, 1989.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.

[KS02] Y. Kalfoglou and M. Schorlemmer. Information-flow-based ontology mapping. *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 1132–1151, 2002.

[KSH12] M. Krötzsch, F. Simancik, and I. Horrocks. A description logic primer. Technical report, University of Oxford, 2012.

[LB87] H.J. Levesque and R.J. Brachman. Expressiveness and Tractability in Knowledge Representation and Reasoning 1. *Computational Intelligence*, 3(1):78–93, 1987.

[Lei94] D. Leivant. Higher order logic. *Handbook of logic in artificial intelligence and logic programming*, 2:229–321, 1994.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[LGW88] D. Lenat, RV Guha, and DV Wallace. The CycL Representation Language. *MCC technival report N ACA-AI-302-88*, 1988.

[LM05] C. Lutz and M. Milicic. A Tableau Algorithm for Description Logics with Concrete Domains and GCIs. *Lecture Notes in Computer Science*, 3702:201, 2005.

[Lor00] R.A. Lorie. Long-term archiving of digital information. 2000.

[Lor01] R.A. Lorie. Long Term Preservation of Digital Information. *Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 346–352, 2001.

[LS+99] O. Lassila, R.R. Swick, et al. Resource Description Framework (RDF) Model and Syntax Specification. 1999.

[Lut03] C. Lutz. Description Logics with Concrete Domains—a Survey. *Advances in Modal Logics*, 4:265–296, 2003.

[LvD05] R. A. Lorie and R. J. van Diessen. Long-term preservation of complex processes. In *Archiving Conference*, volume 2005, pages 14–19. Society for Imaging Science and Technology, 2005.

[LWW07] C. Lutz, D. Walther, and F. Wolter. Conservative Extensions in Expressive Description Logics. *Proc. of IJCAI*, 2007:453–458, 2007.

[MB04] A. Malhotra and P. V. Biron. XML schema part 2: Datatypes second edition. W3C recommendation, W3C, October 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[McG13] G. McGath. The format registry problem. *Code4Lib Journal*, 19, 2013.

[Mei11] C. Meilicke. *Alignment incoherence in ontology matching*. PhD thesis, PhD Thesis, University of Mannheim, Chair of Artificial Intelligence, 2011.

[MGMR02] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: a Versatile Graph Matching Algorithm and its Application to Schema Matching. *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117–128, 2002.

[Min74] M. Minsky. A Framework for Representing Knowledge. Technical report, Massachusetts Institute of Technology, 1974.

[Mit97] T.M. Mitchell. Machine learning. *Burr Ridge, IL: McGraw Hill*, 1997.

[Mot06] B. Motik. *Reasoning in description logics using resolution and deductive databases*. PhD thesis, Universität Karlsruhe, 2006.

[MPSH08] B. Motik, P.F. Patel-Schneider, and I. Horrocks. OWL2 Web Ontology Language: Structural Specification and Functional-Style Syntax. *W3C Working Draft, W3C*, 2008.

[MRG+05] P. Maniatis, M. Roussopoulos, TJ Giuli, D.S.H. Rosenthal, and M. Baker. The LOCKSS Peer-To-Peer Digital Preservation System. *ACM Transactions on Computer Systems*, 23(1):2–50, 2005.

[MSS05] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.

[MST07] C. Meilicke, H. Stuckenschmidt, and A. Tamilin. Repairing Ontology Mappings. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 22, page 1408. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[MSZT+12] C. Meilicke, O. Sváb-Zamazal, C. Trojahn, E. Jiménez-Ruiz, J. L. Aguirre, H. Stuckenschmidt, and B. Grau. Evaluating Ontology Matching Systems on Large, Multilingual and Real-world Test

Cases. Technical report, University of Mannheim, University of Economics Prague, INRIA and LIG Grenoble, University of Oxford, 2012.

[Mug91] S. Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

[MvH04] D. L. McGuinness and F. van Harmelen. OWL web ontology language overview. W3C recommendation, W3C, February 2004.

[NB12] D. Ngo and Z. Bellahsene. YAM++: A multi-strategy based approach for ontology matching task. In *Knowledge Engineering and Knowledge Management*, pages 421–425. Springer, 2012.

[Neb91] B. Nebel. Terminological cycles: Semantics and computational properties. *Principles of Semantic Networks*, pages 331–361, 1991.

[NNS11] J. Noessner, M. Niepert, and H. Stuckenschmidt. Coherent top-k ontology alignment for OWL EL. In *International Conference on Scalable Uncertainty Management*, volume 6929/2011 of *Lecture Notes in Computer Science*, pages 415–427. Springer, 2011.

[OCE08] M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of query answering in expressive description logics via tableaux. *Journal of Automated Reasoning*, 41(1):61–98, 2008.

[OCRMGR15] L. Otero-Cerdeira, F. J. Rodríguez-Martínez, and A. Gómez-Rodríguez. Ontology matching: A literature review. *Expert Systems with Applications*, 42(2):949–971, 2015.

[Par02] R.C. Paré. From Ternary Relationship to Relational Tables: a Case Against Common Beliefs. *ACM SIGMOD Record*, 31(2):46–49, 2002.

[PDF08] Document Management – Portable Document Format – Part 1: PDF 1.7, 2008.

[PPSM09] B. Parsia, P.F. Patel-Schneider, and B. Motik. OWL 2 web ontology language XML serialization. W3C recommendation, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/.

[PSCC10] C. Pesquita, C. Stroe, I.F. Cruz, and F.M. Couto. BLOOMS on AgreementMaker: Results for OAEI 2010. *Ontology Matching*, page 134, 2010.

[RDB+08] J. Ressler, M. Dean, E. Benson, E. Dorner, and C. Morris. Application of Ontology Translation. *Lecture Notes in Computer Science*, 4825/2008:830–842, 2008.

[RDH+04] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. *Engineering Knowledge in the Age of the SemanticWeb*, pages 63–81, 2004.

[Rei87] R. Reiter. *On Closed World Data Bases*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1987.

[Res99] P. Resnick. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. *Journal of Artificial Intelligence*, 11(11):95–130, 1999.

[Ric56] RH Richens. General Programme for Mechanical Translation between Any Two Languages via an Algebraic Interlingua. *2nd International Conference on Mechanical Translation*, 3(2), Oct 1956.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[RMŠZS09] D. Ritze, C. Meilicke, O. Šváb-Zamazal, and H. Stuckenschmidt. A Pattern-Based Ontology Matching Approach for Detecting Complex Correspondences. *The Fourth International Workshop on Ontology Matching*, 2009.

[RNC+95] S.J. Russell, P. Norvig, J.F. Canny, J. Malik, and D.D. Edwards. *Artificial Intelligence: a Modern Approach*. Prentice hall Englewood Cliffs, NJ, 1995.

[Ros10] D.S.H. Rosenthal. Keeping bits safe: how hard can it be? *Communications of the ACM*, 53(11):47–55, 2010.

[RP11] D. Ritze and H. Paulheim. Towards an automatic parameterization of ontology matching tools based on example mappings. In *Proc. 6th ISWC ontology matching workshop (OM), Bonn (DE)*, pages 37–48, 2011.

[RVMŠZ10] D. Ritze, J. Völker, C. Meilicke, and O. Šváb-Zamazal. Linguistic Analysis for Complex Ontology Matching. *Ontology Matching*, page 1, 2010.

[SA08] M.H. Seddiqui and M. Aono. Alignment Results of Anchor-Flood Algorithm for OAEI-2008. In *The 7th International Semantic Web Conference*, page 120, 2008.

[SCC+97] K. A. Spackman, K. EE. Campbell, R.A. CÃ, et al. SNOMED RT: a reference terminology for health care. In *Proceedings of the AMIA annual fall symposium*, page 640. American Medical Informatics Association, 1997.

[Sch91] K. Schild. A correspondence theory for terminological logics: Preliminary report. Technical report, Technische Universität Berlin, 1991.

[Sch96] K. Schild. *Querying knowledge and data bases by a universal description logic with recursion*. PhD thesis, Universität des Saarlandes, 1996.

[Sch09] F. Scharffe. *Correspondence Patterns Representation*. PhD thesis, University of Innsbruck, 2009.

[SE08] P. Shvaiko and J. Euzenat. Ten Challenges for Ontology Matching. *On the Move to Meaningful Internet Systems: OTM 2008*, pages 1164–1182, 2008.

[SEH+03] G. Stumme, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, et al. The Karlsruhe View on Ontologies. *University of Karlsruhe, Institute AIFB, Technical report*, 2003.

[SGGAP14] G. Sidorov, A. Gelbukh, H. Gómez-Adorno, and D. Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, 18(3):491–504, 2014.

[SJ13] P. Shvaiko and Euzenat J. Ontology matching: state of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 2013.

[SMH08] R. Shearer, B. Motik, and I. Horrocks. HermiT: A highly-efficient OWL reasoner. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, pages 26–27, 2008.

[Smu95] R. M. Smullyan. *First-order logic*. Dover Publications, 1995.

[SMW15] Y. Suna, L. Maa, and S. Wangb. A Comparative Evaluation of String Similarity Metrics for Ontology Alignment. *Journal of Information & Computational Science*, 2015.

[Sow91] J.F. Sowa. *Principles of Semantic networks*. Citeseer, 1991.

[Sow00] J.F. Sowa. *Knowledge representation: Logical, Philosophical, and Computational Foundations*. MIT Press, 2000.

[SP04] E. Sirin and B. Parsia. Pellet: An OWL-DL Reasoner. In *2004 International Workshop on Description Logics*, page 212. Citeseer, 2004.

[SPM08] H. Stuckenschmidt, L. Predoiu, and C. Meilicke. Learning Complex Ontology Alignments–A Challenge for ILP Research. *International Conference on Inductive Logic Programming (ILP2008)*, 2008.

[SS88] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431. Morgan Kaufmann, 1988.

[SSS91] M. Schmidt-Schauß and G. Smolka. Attributive Concept Descriptions with Complements. *Artif. Intell.*, 48(1):1–26, 1991.

[ST05] L. Serafini and A. Tamilin. Drago: Distributed Reasoning Architecture for the Semantic Web. *The Semantic Web: Research and Applications*, 3532/2005:361–376, 2005.

[Sta05] A. Stanescu. Assessing the Durability of Formats in a Digital Preservation Environment. *Perspectives*, 21(1):61–81, 2005.

[TB07a] T. Triebsees and U. Borghoff. Towards Automatic Document Migration: Semantic Preservation of Embedded Queries. In *DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering*, pages 209–218, New York, NY, USA, 2007. ACM.

[TB07b] T. Triebsees and U.M. Borghoff. A Theory for Model-Based Transformation Applied to Computer-Supported Preservation in Digital Archives. *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 359–370, 2007.

[TH04] D. Tsarkov and I. Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, volume 104, pages 41–50, 2004.

[TH06] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. *Automated Reasoning*, pages 292–297, 2006.

[THPS07] D. Tsarkov, I. Horrocks, and P.F. Patel-Schneider. Optimizing terminological reasoning for expressive description logics. *Journal of Automated Reasoning*, 39(3):277–316, 2007.

[Tob01] S. Tobies. *Complexity results and practical algorithms for logics in knowledge representation*. PhD thesis, RWTH Aachen, 2001.

[TP10] E. Thomas and Y. Pan, J. Z.and Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In *The Semantic Web: Research and Applications*, pages 431–435. Springer, 2010.

[TYK12] Y. Tzitzikas, Marketakis Y., and Y. Kargakis. Conversion and emulation-aware dependency reasoning for curation services. *Proceedings of the 9th Annual International Conference on Digital Preservation (iPres2012), Oct. 2012, Toronto.*, 2012.

[Var82] M. Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

[vdHvDvdM05] JR van der Hoeven, RJ van Diessen, and K. van der Meer. Development of a Universal Virtual Computer (UVC) for long-term preservation of digital objects. *Journal of Information Science*, 31(3):196, 2005.

[vdHVW05] JR van der Hoeven and HN Van Wijngaarden. Modular emulation as a long-term preservation strategy for digital objects. *Proceedings of the 5th International Web Archiving Workshop (IWAW05) Held in Conjunction with the 8th European Conference on Research and Advanced Technologies for Digital Libraries (ECDL 2005), September*, pages 22–23, 2005.

[VL04] L. Van Lier. *The ecology and semiotics of language learning: A sociocultural perspective*, volume 3. Springer Science & Business Media, 2004.

[VVR+05] G. Varelas, E. Voutsakis, P. Raftopoulou, E.G.M. Petrakis, and E.E. Milios. Semantic Similarity Methods in WordNET and their Application to Information Retrieval on the Web. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 10–16. ACM, 2005.

[vZSS09] O. Šváb Zamazal, V. Svátek, and F. Scharffe. Pattern-based Ontology Transformation Service. In *Proceedings of the 1st International Conference on Knowledge Engineering and Ontology Development*, 2009.

[Wei08] M. Weiten. OntoSTUDIO® as a Ontology Engineering Environment. *Semantic Knowledge Management: Integrating Ontology Management, Knowledge Discovery, and Human Language Technologies*, page 51, 2008.

[WGCN12] S. N. Wrigley, R. García-Castro, and L. Nixon. Semantic evaluation at large scale (SEALS). In *Proceedings of the 21st international conference companion on World Wide Web*, pages 299–302. ACM, 2012.

[Wol08] L. A. Wolsey. Mixed integer programming. *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[Yeo10] G. Yeo. 'Nothing is the same as something else': Significant Properties and Notions of Identity and Originality. *Archival Science*, pages 1–32, 2010.

University of Bamberg Press

Being able to read successfully the bits and bytes stored inside a digital archive does not necessarily mean we are able to extract meaningful information from an archived digital document. If information about the format of a stored document is not available, the contents of the document are essentially lost. One solution to the problem is format conversion, but due to the amount of documents and formats involved, manual conversion of archived documents is usually impractical. There is thus an open research question to discover suitable technologies to transform existing documents into new document formats and to determine the constraints within which these technologies can be applied successfully.

In the present work, it is assumed that stored documents are represented as formal description logic ontologies. This makes it possible to view the translation of document formats as an application of ontology matching, an area for which many methods and algorithms have been developed over the recent years. With very few exceptions, however, current ontology matchers are limited to element-level correspondences matching concepts against concepts, roles against roles, and individuals against individuals. Such simple correspondences are insufficient to describe mappings between complex digital documents.

This thesis presents a method to refine simple correspondences into more complex ones in a heuristic fashion utilizing a modified form of description logic tableau reasoning. The refinement process uses a model-based representation of correspondences. Building on the formal semantics, the process also includes methods to avoid the generation of inconsistent or incoherent correspondences.

In a second part, this thesis also makes use of the model-based representation to determine the best set of correspondences between two ontologies. The developed similarity measures make use of semantic information from both description logic tableau reasoning as well as from the refinement process. The result is a new method to semi-automatically derive complex correspondences between description logic ontologies tailored but not limited to the context of format migration.

eISBN: 978-3-86309-578-9

9 783863 095789

www.uni-bamberg.de/ubp