Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

Change Management in the Traditional and Semantic Web

by

Alessandro Solimando

Theses Series **DIBRIS-TH-2015-05**

DIBRIS, Università di Genova
Via Opera Pia, 13 16145 Genova, Italy `http://www.dibris.unige.it/`

Università degli Studi di Genova

Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

Change Management in the Traditional and Semantic Web

by

Alessandro Solimando

February, 2015

Dottorato di Ricerca in Informatica
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università degli Studi di Genova

DIBRIS, Università di Genova
Via Opera Pia, 13
I-16145 Genova, Italy
http://www.dibris.unige.it/

Ph.D. Thesis in Computer Science (S.S.D. INF/01)

Submitted by Alessandro Solimando
DIBRIS, Università di Genova
alessandro.solimando@unige.it

Date of submission: February 2015

Title: Change Management in the Traditional and Semantic Web

Advisors:
Giovanna Guerrini
Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
Università di Genova, Italy
giovanna.guerrini@unige.it

Ernesto Jiménez-Ruiz
Computer Science Department
University of Oxford, United Kingdom
ernesto.jimenez-ruiz@cs.ox.ac.uk

External Reviewers:
Rafael Berlanga Llavori
Departament de Llenguatges i Sistemes Informàtics
Universitat Jaume I de Castelló, Spain
berlanga@uji.es

Riccardo Rosati
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Università di Roma La Sapienza, Italy
rosati@dis.uniroma1.it

Abstract

Data has played such a crucial role in the development of modern society that relational databases, the most popular data management systems, have been defined as the "foundation of western civilization" for their massive adoption in business, government and education, that allowed to reach the necessary productivity and standardization rate.

Data, however, in order to become knowledge, needs to be organized in a way that enables the retrieval of relevant information and data analysis, for becoming a real asset. Another fundamental aspect for an effective data management is the full support for changes at the data and metadata level. Data is, nowadays, archived and kept as a strategic resource, by private entities as well as corporation and public administrations, but a simple archival system is not enough to preserve the capability of querying and manipulating the stored information. A failure in the management of data changes usually results in a dramatic dimishment of its usefulness. Data does evolve, in its format and its content, due to changes in the modeled domain, error correction, different required level of granularity, in order to accomodate new information.

The red thread that links together all the contributions presented in this thesis is the concept of change management. Due to the vastity of the topic, and to the impressive number of publications covering its different aspects, it is unconceivable to present a single framework tackling all the aspects of the problem. Concretely, we investigated relevant change management problems in the context of the so called World Wide Web.

The first half of the thesis deals with XML, the W3C-endorsed and widely used markup language to define semi-structured data, proposed in order to represent and exchange data (not only in the Web context), overcoming the limitations of the unstructured and purely syntactical management of Web data. Specifically, we first investigate the static analysis of document updates aiming at the restoration of document validity w.r.t. an updated schema. The proposed technique models the schema as a Hedge Automaton (automaton for unranked trees), documents as trees, and the

document update operations as a hedge rewriting system. A second investigated issue is the synthesis of edit-scripts, transforming a source XML document into an updated one, expressed using XQuery Pending Update Lists, key ingredient for any versioning system.

Favored by the tight relationship between documents and schemas, XML change management has been conceived as an holistic process, involving all the different data, and related metadata, as a whole, and the mutual impacts of changes over any of the components of a data management environment. This research area, referred to as XML co-evolution, deepened the effects of updates involving document to the associated schema and the other documents in the same data collection, as well as the impact of schema modifications over the documents conformant to it.

The second half of the thesis considers one of the enabling factors of the Semantic Web vision, ontologies and related metadata (ontology mappings, in particular). By means of ontologies, the computers do not really understand meanings, but they are allowed to manipulate intended semantics in an unambiguous (that is, logical) and consistent way. But in order to be compatible with a deeply decentralized system as the Web, the semantic extension enabled by means of ontologies needs to be decentralized as well. The problem of (semi-)automatically computing mappings between independently developed ontologies is usually referred to as the ontology matching problem.

Due to the precise logical semantics associated with ontologies and their mappings, ontological change management cannot operate at the syntactical level only, ignoring logical consequences. Therefore, debugging techniques for unintended consequences arising in the context of ontology matching need to be considered as a fundamental aspect, for change management as well. In addition, ontologies, as any other conceptual schema, are constantly evolving, and due to their complexity, they usually benefit from an incremental development, blending together ontology data and metadata change management, ontology development and debugging techniques.

I dedicate this dissertation work to all the people which supported me throughout these years.

A special mention goes to my family, to my beloved Susanna, to my friends Davide, Luca, Paolo, Riccardo and Valentina.

Hearthfelt thanks to Federico, Fabio, Valerio, André, Jesús and Federico, good friends and not only colleagues.

Acknowledgements

I want to thank the following people who deeply contributed to the present work and to my professional growth during these years:

- my supervisors, for their invaluable advices and support
- my co-authors, in particular Federico Cavalieri and Giorgio Delzanno
- Marco Maratea for his help on Answer Set Programming
- the DASI lab at the "La Sapienza" University of Roma, for the discussions and brainstormings during my visits
- Viviana Mascardi for her interest and feedbacks on my work
- Christian Meilicke and Heiner Stuckenschmidt for sharing their experience about ontology alignment debugging

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the thesis we address the problem of change management on (part of the main ingredients of) the World Wide Web. In particular, we focus our attention on semi-structured data and logic-based data management, the latter at the basis of the Semantic Web. In this introductory chapter we first discuss context and motivations for our work in Section 1.1, while a deeper introduction for the two main cores of the thesis is given in Section 1.2 and Section 1.3, respectively. Finally, the contributions of the thesis are detailed in Section 1.4, that also provides an outline of the remainder of the document.

1.1 Context and Motivations

Data has played such a crucial role in the development of modern society that relational databases, the most popular data management systems, have been defined as the "foundation of western civilization" [Win05] for their massive adoption in business, government, and education, that allowed to reach the necessary productivity and standardization rate.

Data, however, in order to become knowledge, needs to be organized in a way that enables the retrieval of relevant information and data analysis, for becoming a real asset. Another fundamental aspect for an effective data management is the full support for changes at the data and metadata level. Data is, nowadays, archived and kept as a strategic resource, by private entities as well as corporation and public administrations, but a simple archival system is not enough to preserve the capability of querying and manipulating the stored information. A failure in the management of data changes usually results in a dramatic dimishment of its usefulness. Data does evolve, in its format and its content, due to changes in the modeled domain, error correction, different required level of granularity, in order to accomodate new information.

The red thread that links together all the contributions presented in this thesis is change man-

agement. Due to the vastity of the topic, and to the impressive amount of work covering its different aspects, it is unconceivable to present a single framework tackling all the aspects of the problem. Concretely, we investigated less addressed (but still relevant) problems, in the context of the so called *World Wide Web* [Tim90] (or simply the Web), that revolutionized our concept of communication.

Since its advent in 1989, an increasing amount of information has been made available through the years, in a way that the Web can be defined as the biggest available distributed database in the world. In such a loosely coupled context, the limits of an unstructured and purely syntactical management of the web data rapidly emerged. *XML* [W3C08], the *EXtensible Markup Language*, a W3C-endorsed and widely used markup language to define semi-structured data, has been proposed, in order to represent and exchange data. XML brought to the Web the capability of structuring data, but with a user-defined set of tags. This feature provides all the benefits of structured information but with the needed flexibility for an inherently distributed environment such as the Web. Furthermore, XML comes with a machine-readable format easy to understand also for human beings. In addition, despite XML documents can be used without a schema (a structural definition of a valid document), many application scenarios benefit from its presence. Even if a schema is provided, the compliance check of a document w.r.t. it (process known as document validation [HM02]) is never mandatory.

Favored by the tight relationship between documents and schemas, change management has been conceived as a holistic process, involving data, and related metadata, as a whole, and the mutual impacts of changes over any of the components of a data management environment. This research area, referred to as co-evolution, in the context of XML deepened the effects of document updates over the other documents sharing the same schema, and the schema itself, as well as the impact of schema modifications over the documents conformant to it. The first part of the thesis relates to the subfield of XML co-evolution.

The evolution of the Web started another phase in 2001, with a vision paper by Tim Berners Lee (the father of the Web), James Handler and Ora Lassila [BLHL01]. In that paper, the term *Semantic Web* has been coined, and the requirements for completing the path towards a more organized and semantical version of the Web, started with the massive employment of XML in serializing and exchancing data, have been formalized. XML allowed to structure data in a flexible manner, through a set of custom tags, those intended semantics was still not possible to be understood by machines. As the adoption of XML did not retire HTML or Web search engines, the advent of Semantic Web is extending the traditional one, allowing to give to information a well-defined meaning, and enabling the cooperation of computers and people. The power of its universal interconnection has also inspired a new data format known as graph data format, where the notion of (labeled) relation mimicks the link between Web pages. *Resource Description Framework* (RDF) [W3C04, W3C14] is a prominent example of graph data format, that has been developed in the context of Semantic Web, where an (arbitrary) relationship between an object and a subject is encoded as a statement in form of a triple, that can be written using

XML, JSON [ECM], etc. Any object or subject is denoted by a *Uniform Resource Identificator* (URI) [IET], of which the *Uniform Resource Locator* is probably the best known instantiation.

The research area, inside the broader area of *Artificial Intelligence* (AI), addressing the problem of assigning a (logical) semantics to data, in order to allow the use of inference rules to deduce new data/facts, and to check information consistency, is known as *Knowledge Representation* (KR). The key ingredient in the development of the semantic layer proper to the Semantic Web is known as ontologies (also called knowledge bases), that is, a shared conceptualization of a domain of interest, as Gruber defined it in a 1993 article [Gru93]. More specifically, ontologies describe a domain as a set of classes (uniform aggregates of individuals sharing a characteristic of interest), and the relationships between classes and individuals.

The first ontology languages proposals, *frames* and *semantic networks*, were lacking of a logic-based semantics [BCM$^+$03]. *Description Logic* (DL) was first introduced into KR systems, during the 80's, to overcome this deficiency [BCM$^+$03]. DL is a family of formal knowledge representation languages, at a varying expressive power and different computational complexity of the associated reasoning tasks. DL has been successfully applied in many diverse fields, ranging from Natural Language Processing (NLP) to Databases, from Software Engineering to the Semantic Web.

In the Semantic Web, each XML tag, for instance, can be associated (that is, annotated) with ontological concepts, enabling the use of inference rules on them, as well as their relationships. The same can occur for information encoded in a web page, leveraging from complex NLP tasks, aiming at extracting the intended meaning from free text, using an unrestricted vocabulary. Another example of use of ontologies on the Web is to improve precision of search engines, allowing to rely on precise concepts instead of possibly ambiguous keywords and strings.

By means of ontologies, computers do not really understand meanings, but they are allowed to manipulate intended semantics in an unambiguous (that is, logical) and consistent way. But in order to be compatible with a deeply decentralized system as the Web, the semantic extension enabled by means of ontologies needs to be decentralized as well.

In such context, it is natural that the same domain of interest is modeled by different institutions, with different aims and possibly using different terminologies. In order to allow interoperability between ontology-based systems and data, ontology matching techniques have been developed during the years. The problem of (semi-)automatically computing mappings between independently developed ontologies is usually referred to as the *ontology matching problem*. Ontology matching systems, taken as input a pair of (input) ontologies, compute a set of correspondences between their entities. A number of sophisticated ontology matching algorithms have been developed in the last years [ES10, EMS$^+$11, SE12]. These systems, however, rely on lexical and structural heuristics and the integration of the input ontologies and the mappings may lead to many undesired logical consequences, that may hinder the usefulness of ontology mappings. The occurrence of these violations is not only frequent in the mappings generated by top-level

matchers, but also in the reference mapping sets of the *Ontology Alignment Evaluation Initiative*[1] (*OAEI*), the most important evaluation in the field. Also manually curated mappings, such as *UMLS-Metathesaurus* [Bod04] (*UMLS*), a comprehensive effort for integrating biomedical knowledge bases, suffer from these violations.

The additional semantic layer realized using ontologies, on one hand unleashes new possibilities, but on the other hand forces to consider specific semantical and logical problems, that were never considered before, due to the mainly syntactical nature of the traditional Web.

For instance, adaptation algorithms for ontology mappings (upon ontology changes), cannot be unaware of logical violations while reflecting these changes, and therefore cannot simply deal with the problem from a syntactical point of view. However, ontologies, as any other conceptual schema, are constantly evolving, and due to their complexity, they usually benefit from an incremental development. This feature blends ontology data and metadata change management into ontology development and management techniques (*e.g.*, [JRCHB11b], where a concurrent ontology development and versioning tool with repair facilities has been proposed). For what concerns ontology mappings, for instance, the debugging capabilities (that is, detecting and correcting logical violations) practically enable change management, despite not directly linked to it, and the same holds for ontology development and maintenance.

The second part of the thesis will therefore address the problem of change management for metadata related to DL ontologies (ontology mappings, in particular), and logical debugging techniques associated to ontology mappings.

1.2 Part I - XML Co-Evolution

Two reasons behind the large adoption of XML, beyond the markup language itself, have been the capability of associating a schematic representation with a document collection, and the query and update languages for conveniently manipulating XML documents.

Different schema definition languages have been proposed (DTD [W3C07], RelaxNG [CM01], XML Schema [W3C12]) and their expressiveness has been formally compared [MLMK05]. The most commonly adopted language XML Schema, that is a W3C recommendation employing an XML representation.

On the query language side, the standards for XML are *XPath* [Jam99, W3C10a] and *XQuery* [W3C10b, W3C13] (subsuming XPath), followed by its extension targeting update primitives, *XQuery Update Facility* [Don09]. XQuery Update has the unique characteristic that each expression can be translated, during the evaluation phase, into a sequence of simple update operations called *Pending Update List* (PUL), that can be then applied to the document in a successive

[1] `http://oaei.ontologymatching.org/`

step. Each PUL operation targets a single node, and given the relative simplicity of this set of basic operations, an algebra for manipulating PULs have been proposed, enabling a set of effective and efficient dynamic optimization techniques [Cav13].

Despite XML has been originally defined as a schemaless document format, the knowledge about document structure, as defined in a schema, has been applied to help both software products and users to define the contents and organize them inside an XML document. For instance, XML schemas have been successfully used in static analysis techniques for optimizing the temporal and spatial requirements of query processing tasks, exploiting the structural definition of a document collection [BBC+11, BCM+13, BCCN13].

Schemas, as any other conceptual model, are mutable, and a schema modification may be needed for many reasons, ranging from changes in the application domain that need to be reflected in the associated schema, to revisions on ongoing or recently developed data representation standards. Error correction is another important motivation for proposing and applying a change.

Two general approaches to schema updates in the schema management context exist, namely schema versioning (where the schema updates produce a new schema version) and schema evolution (where the updated schema replaces the previous schema version). The effects of schema updates are not constrained at the schema level, but they have an impact on the associated document collection, whose validity is usually affected.

Different techniques for supporting schema updates have been proposed in literature [CGM11b, CGM14]. The problem of restoring document validity after schema changes has been addressed in three ways: *(i)* documents are left untouched, *(ii)* a (minimal) document adaptation is automatically proposed, *(iii)* a manual document adaptation is proposed by the user. A document adaptation here is a set of document updates having the goal of restoring the conformance to the schema. For both stategies updating the documents, a revalidation step for each document is needed, and given that the size of a document collection can be significant (both in terms of single document size and the number of involved documents), the impact of a schema update can be highly demanding. Given that, as already mentioned, XML tags have semantic information associated with the modeled data which is not encoded at a machine readable level, the semantics of a schema update cannot be automatically captured. In practice, the corresponding document adaptation often needs to be defined by the user. Therefore the third option is sometimes the only viable choice.

In the thesis we will propose a technique that does not only accomodate user-proposed adaptations but, following a static analysis approach, it could also avoid the document revalidation phase for the whole document collection, thus in principle mitigating the aforementioned limitation. The proposed approach exploits the well known relationship between schemas and tree automata [MLMK05], and translates the original and updated schemas (S and S", respectively) into the corresponding automata A and A". Then, the proposed document adaptation, expressed using PULs (with some limitations), is translated into a set of tree rewriting operations, and the

algorithm computes the associated modifications to the transitions of the original automaton A, thus computing the modified automaton A'. This transformation can be conceived as a relation linking original and adapted documents (through the transformation itself). The domain of such relation is the original document collection, accepted by automaton A, while its image is the adapted document collection, accepted by the automaton A'.

Then, if the language accepted by A' is contained by that accepted by A", we are guaranteed (without the need of any runtime revalidation step), that any document valid w.r.t. the original schema S, once adapted using the proposed document adaptation, will result in a document accepted by A' (and therefore also by A", due to the inclusion). And, by construction, any document accepted by A" is necessarily conformant w.r.t. the updated schema S".

As already mentioned, XML evolution does not only take into account schema updates, but also addresses updates at the document level. The capability of detecting and representing document changes in a compact way has always been crucial in the context of data management. The algorithms responsible for this task are usually called *differencing algorithms* and, taken as input a pair of documents, they produce as output a set of update operations (in a given language) that can transform the first document into the second one. These sets of operations usually follow a minimality criterion, and are commonly referred to as *edit-scripts* or *deltas*. Edit-scripts are a powerful tool in change management systems for analysing document transformations and enabling practical document versioning systems.

The thesis proposes a differencing algorithm which is the first proposal addressing PULs as a language for expressing edit-scripts. We find this characteristic quite significant because PULs: *(i)* are simple (edit-scripts can be easily understood); *(ii)* can be efficiently reverted, combined and minimized; *(iii)* their application does not require ad-hoc engines, but any XQuery Update compliant one can be used; *(iv)* are a formal update representation that enables query independency analysis (*i.e.*, affected indexes and materialized views can be identified) *(v)* enable reasoning over interesting properties like commutativity, conflict detection and merging, and can therefore be used for reconciling off-line updates over the same document in a collaborative environment.

The proposed algorithm improved the state of the art from both the performance and generated edit-script quality perspectives.

1.3 Part II - Ontology Alignment Debugging and Evolution

Ontologies play a key role in the development of the Semantic Web and are being used in many diverse application domains. An application domain may have been modeled according to different points of view and purposes. This situation usually leads to the development of different ontologies that intuitively overlap, but that use different naming and modeling conventions. The

mappings computed by ontology matching systems may lead to many undesired logical consequences in the aligned ontology.

Ontological knowledge bases are not only relevant in the pure Semantic Web context, but are nowadays widely employed in many different scenarios for their unique combination of nice computational properties and good expressive power for modelling purposes. Such ontologies are often shared by different institutions and metadata are defined on top of them in order to facilitate their discovery, reuse, profiling and sharing. A prominent example in the biomedical field is represented by *BioPortal*, that is the most comprehensive effort for collecting, sharing and mapping biomedical ontologies [MNS+11, WNS+11]. Due to the size and complexity of such ontologies, as well as their high update rate, the need for adaptation techniques for the (semi-)automatically maintenance of related metadata upon ontology updates clearly emerges.

Specifically, the problem of proposing ontology mappings adaptations has been extensively studied in literature [MS09, DRPDSRD12, DRDP+13, GDRH+13, RSD+14, DDRP+14]. Due to the precise semantics of the underlying ontologies, the logical consequences imposed by any modification to the data and related metadata, should be taken into account.

The relationship between ontology mapping adaptation and ontology mapping debugging fields, carried out in the present thesis, is a motivation for a holistic approach. In particular, the thesis also discusses and analyzes the drawbacks of the main ontology mapping adaptation approaches, agnostic of the logical layer.

In the context of ontology mappings, three principles were proposed to minimize the number of potentially unintended entailments and to capture logical consequences [JRCHB11a], namely: *(i) consistency principle*, the mappings should not lead to unsatisfiable classes in the integrated ontology, *(ii) locality principle*, the mappings should link entities that have similar *neighbourhoods*, *(iii) conservativity principle*, the mappings should not introduce new semantic relationships between concepts from one of the input ontologies. Violations to these principles may hinder the usefulness of ontology mappings in practical contexts such as query answering or data integration.

Despite the increasing number of contributions addressing ontology alignment debugging, the conservativity principle has received little attention. A possible explanation is that the negative effects of violations to the consistency principle (that is, incoherences and inconsistencies) are already evident for any ontology alignment application scenario, and therefore were considered at an earlier stage.

Addressing the conservativity principle violation requires both a detection and a repair technique. For violation detection, the thesis proposes a complete technique, based on an efficient interval labelling schema [ABJ89] for the input/aligned ontologies.

For conservativity violations affecting atomic concepts not involved in a subsumption relationship nor sharing any descendant, the problem can be reduced to a consistency repair by in-

serting a disjointness axiom between the two concepts. A classic approach for debugging ontologies is to compute a repair by computing a (minimal) *hitting set* over the set of justifications [KPHS07, HPS08, HPS10] (minimal sets of axioms entailing a consequence). Computing all the justifications for a given entailment is a costly reasoning service, and all the scalable debugging algorithms propose approximate repair computations [JRMCH13, Mei11, NB12, PFSC13]. To address the scalability problem when dealing with large ontologies and mapping sets, our method actually relies on the (Horn) propositional projection of the input ontologies, but does not ensure completeness. To this aim we have adapted the infrastructure provided by *LogMap* matcher [JRC11, JRCZH12].

For the violations affecting concepts already involved in a subsumption relationship in the input ontologies, by contrast, a graph representation is used, exploiting the property that part of these violations forms a cycle (one half represents the previous subsumption relationship, the other one representing the violation). The detection and repair strategies work on the strongly connected components (SCCs) of the graph representation of the aligned ontology, exploting the well-known relation between SCCs and directed cycles. The approximate repair aims at removing all the cycles corresponding to a violation by computing a solution to an ad-hoc variant of the *Feedback Edge Set* problem [ENSS98], encoded as a logic program.

1.4 Thesis Contributions and Outline

In this section we highlight the contributions of the different chapters composing the thesis, while outlining its structure.

Chapter 2 – Static Analysis of XML Schema Document Adaptations

The content of this chapter has first been published in [SDG12b, SDG12a], then accepted in an extended version by the *Theoretical Computer Science* journal [SDG14], and presents the following contributions.

First, we have provided a complete formal definition of the parallel semantics (Section 2.5), and of the automata transformations for PUL updates, as well as a stronger notion of Hedge Automaton normalization w.r.t. the one existing in literature (Section 2.6).

Furthermore, we have given a correctness proof of the proposed framework. The complexity of the algorithm has been investigated in the worst-case scenario and a more realistic one (Section 2.6).

Finally, we have added experimental results of a prototypical implementation by considering different parameters to evaluate the cost of the automata transformation and of the inclusion

test. The hypotheses underpinning our complexity for realistic schemas and updates have been empirically validated using a well established dataset in the XML context (Section 2.7).

Chapter 3 – Synthetising Changes in XML Document as Pending Update Lists

The work presented in this chapter has been published in [CSG13a] and presented at *Very Large Data Bases* 2014 conference. An extended accompanying version of the experimental results has been published as a technical report [CSG13b]. An additional extended version, including the aforementioned ones, was published in the PhD thesis of Federico Cavalieri [Cav13], one of the co-author of the joint work.

The main contributions inside our algorithm are: *(i)* the first proposal expressing edit-scripts as PULs, a formally grounded W3C standard, which can be evaluated by existing XQuery Update Facility engines and effectively manipulated [Cav13, CGM11a]; *(ii)* improvement of the state of the art with respect to edit-script quality and differencing time, as experimentally verified in Section 3.7; *(iii)* an efficient tree edit-distance estimator based on pq-grams that showed low bias when used for estimating the minimal PUL transforming a tree into another (Section 3.6); *(iv)* an exact and optimal solution to the problem of finding, given a set of two-dimensional weighted points, one of the maximal-weight subsets defining a strictly increasing function; the log-linear algorithm generalizes an existing solution for finding heaviest increasing common subsequences [JV92] (Section 3.5); *(v)* and an extensive experimental evaluation of the main contribution and all of the employed heuristics (Section 3.7).

Chapter 4 – Ontology Matching: Basics and Motivations

This chapter provides the necessary preliminaries and motivations for the second part of the thesis. The novel contributions are limited to Section 4.7, and are as follows.

In order to study more advanced evaluation metrics for ontology-to-ontology alignments (w.r.t. the comparison against a reference alignment), we proposed a novel evaluation track inside OAEI. The aim of this track is to evaluate ontology alignments w.r.t. their capability of enabling query answering, in a context in which multiple ontologies exist. The track also addresses the role of conservativity and consistency violations, as well as their repair techniques. The novel *OA4QA* track has been integrated into the 2014 edition of the OAEI, and the main ideas and results have been published in [SJP14, DEE+14].

Another contribution of this chapter, relates to the first study concerning the relationships and cross-fertilization between the fields of ontology matching and ontology (alignment) evolution, and has been accepted at the *Doctoral Consortium* of the *International Semantic Web Conference*

2014, and then selected for being published in the proceedings volume of the conference [Sol14]. In this paper, the main goals and hypotheses underlying the study of conservativity violations in ontology alignments, and the study of the relationships between ontology alignment debugging and evolution, have been presented.

Chapter 5 – Equivalence Conservativity Principle Violations

This chapter focuses on a subset of the conservativity violations, resulting in novel equivalences between named classes, entailed by the aligned ontology (not holding in the input ontologies).

The contributions of the chapter can be summarized as follows: *(i)* a formal definition of equivalence violations regarding the taxonomical skeleton of ontologies in terms of graph-theoretical notions (Section 5.3), *(ii)* a characterization of repair for such violations using a variant of the well-known *NP*-hard *Weighted Feedback Edge Set* problem (Section 5.3), *(iii)* definition of a debugging algorithm, in two flavors, to automatically detect violations and to compute a repair based on logic programming (*i.e.*, *Answer Set Programming*), and *(iv)* its temporal complexity analysis (Section 5.4). In addition, *(v)* an extensive experimental evaluation of the proposed approaches is provided, as well as *(vi)* an analysis of the results of the *OAEI* 2012–2014 campaign, with an implicit comparison with the state of the art debugging tools and ontology matchers (Section 5.5).

The main ideas of the chapter (that is, the formalization of equivalence violations using graph-theory, their detection using strongly connected components and the diagnosis computation using logic programming) have been published in [SJG14].

Chapter 6 – Subsumption Conservativity Principle Violations

This chapter tries to bridge the gap between the proposal of the previous chapter, and a full detection and repair algorithm for conservativity violations.

The approach presented in this chapter, instead of characterizing violations by means of a graph-theoretical concepts, aims at (partially) reducing subsumption violations repair, to a consistency repair problem.

The fundamental idea underlying this approach, that is enriching the input ontologies with additional disjointness axioms, is not novel in literature. However, the aspects of novelty consist in: *(i)* the first method addressing the conservativity principle with a theoretical foundation of the concrete ontology fragment covered by both the detection and repair techniques (Section 6.3); *(ii)* the definition of direct and derived subsumption violations, for facilitating user-interaction and presentation of the results to the domain experts (Section 6.3); *(iii)* an automatic and efficient identification and addition of a small set of disjointness axioms, using interval indexing

(Section 6.4); *(iv)* an approximate notion of subsumption conservativity violations that allows the reuse of existing repair techniques for the consistency principle, and that allows to solve, in practice, almost all the conservativity violations, considering the full notion (Section 6.4). *(v)* an extensive experimental evaluation of the different alternative algorithms is provided, as well as *(vi)* an analysis of the results of the *OAEI* 2012–2014 campaign (Section 6.5).

The main contributions of the chapter have been published in [SJRG14], but they have been extended in the thesis with the missing proofs and more experimental results.

Chapter 7 – A Combined Approach for Conservativity Principle Violations

Finally, this chapter combines, in an appropriate way, the best variants of the approaches presented in the previous chapters, in order to fully address the detection and repair of the conservativity principle violations, in practice.

The combined approach is presented in two flavors, which apply the single repair algorithms in a different ordering (Section 7.2).

The hypothesis of an increased effectiveness of the multi-strategy approach has been experimentally verified over a dataset composed by the alignments computed by the participating systems at the *OAEI* campaign in the years 2012–2014.

By means of the aforementioned dataset, we also experimentally evaluated that the efficiency is comparable with that of the single repair algorithms (Section 7.3).

The main contribution of this chapter, that is the combination of the two approaches targeting subsumption and equivalence violations, in a single multi-strategy technique, has been published in [SJG14].

The existing paper has been integrated with the detailed definition of the algorithm, more experimental results and a practical comparison of the two variants.

Part I

XML Co-Evolution

Chapter 2

Static Analysis of XML Schema Document Adaptations

2.1 Introduction

XML is a widely employed standard for the representation and exchange of data on the Web. XML does not define a fixed set of tags, and can thus be used in a great variety of domains. The structure of an XML document can be optionally specified by means of a schema, expressed as an XML Schema [Pri04] or as a DTD [W3S], and the document structural information can be exploited for efficient document handling. A given schema can be used by different users to locally store documents valid w.r.t. the schema. In a dynamic and heterogeneous world as the Web, updates to such shared schemas are quite frequent and a support for dynamic schema management is crucial to avoid a diminishment of the role of schemas. As a consequence of a schema update, document validity might need to be re-established and no automatic way to adapt documents to the new schema may exist, since the adaptation may require knowledge of the element semantics.

In the context of XML schema updates, the problem we consider in this chapter is defined as follows. Given two XML schemas S and T, and a sequence of document updates υ (*i.e.*, the "adaptation"), are the documents obtained by applying υ to the instances of S compliant to T? The challenge is to prove that υ is correct without having to revalidate each instance of S.

We can motivate the problem with the help of two examples. We first assume that different administrators share a common schema containing an optional element `address`. Assume now that the schema is updated by inserting `zipcode` as a sibling of `address` (optional sequence). The first administrator can automatically retrieve all the zipcode data, and insert them to adapt the documents. The other administrator, instead, cannot easily obtain these data, and prefers to

restore the document validity by deleting each address, an operation that does not directly correspond to the one occurred on the schema (*i.e.*, an insertion). We now consider a second scenario in which only the original schema S and the target schema T are known, while the schema update sequence is not. Individual administrators may thus specify document adaptations, intended to transform any document valid for S into a document valid for T. In both examples, methods that are able to validate the specified document adaptations are useful to avoid the expensive run-time revalidation of documents resulting from the application of such adaptations.

In this chapter we present an automata-based method, called HASA (Hedge Automata Static Analysis) module, for the static analysis of XML document adaptations, expressed as sequences of *XQuery Update Facility* (XQUF) [Don09] update primitives. Hedge Automata, on top of which our proposal is built, are a very flexible and general tool for manipulating trees. Validation algorithms for XML schemas are naturally expressed via Hedge Automata. It is also well-known that any XML schema can be represented through a corresponding Hedge Automaton [MLMK05]. An XML Schema can be viewed as an extended grammar that finitely describes a (finite or infinite) collection $\mathcal{D}$ of XML documents. An Hedge Automaton associated with a given XML Schema is an extended finite automaton that recognizes a term representation of any document in $\mathcal{D}$. Based on this, the key feature of HASA is the use of an automatic inference method for extracting the type of a sequence of document updates. The type is computed starting from the static type extracted from an XML schema, and from rewriting rules that formally define the operational semantics of a sequence of document updates. Type inclusion can then be used as a conformance test w.r.t. the type extracted from the updated XML schema.

Outline. The chapter is organized as follows. Section 2.2 discusses related work, Section 2.3 introduces preliminary definitions, while in Section 2.4 Hedge Automata are defined as a formal representation of XML schemas. In Section 2.5 we introduce a parallel rewriting semantics for modelling the effect of a document update on a term-based representation of XML documents. Our semantics is based on a representation of document updates as special types of term rewriting systems [JR10], and on a parallel semantics for modeling the simultaneous application of a rewrite rule to each node that satisfies its enabling conditions (we consider here label-based node selection only). In Section 2.6 we define a symbolic algorithm to compute the automaton that recognizes the adapted documents as a Hedge Automata transformation, and we prove its correctness w.r.t. our parallel rewrite semantics. This operation, called $Post$, is the core operation of our HASA approach. It is important to remark that, differently from other automata-based transformation approaches [Tou12], we compute the effect of a single document update and not of its transitive closure. Section 2.7 describes the experimental evaluation of our technique.

2.2 Related Work

Concerning related work on static analysis, the main formalization of schema updates is represented by [BC09], where the authors take into account a subset of XQUF which deals with structural conditions imposed by tags only. Type inference for XQUF, without approximations, is not always possible. This follows from the fact that modifications that can be produced using this language may lead to nonregular schemas, that cannot be captured with existing schema languages for XML. This is the reason why [BC09], as well as [Tou12], computes an over-approximation of the type set resulting from the updates. In our work, on the contrary, to produce an exact computation, we are forced to cover a smaller subset of XQuery Update features: [BC09], indeed, allows the use of *XPath* axes to query and select nodes, with a mixture of selectivity conditions and positional constraints in the pattern to match. In our work, as well as in [JR10] and [Tou12], we have considered update primitives only, thus excluding complex expressions such as "for loops" and "if statements", based on the result of a query. These expressions, anyway, may be translated into a sequence of primitive operations: an expression using a "for loop", for instance, repeats n times a certain primitive operation, and therefore can be simulated with a sequence of n instances of that single primitive operation.[1] However, tests for loops and conditional statements based on query results over documents are of course not expressible working only at schema level.

Macro Tree Transducers (MTT) [MBPS05] can also be applied to model XML updates as in the *Transformation Language* (TL), based on *Monadic Second-Order logic* (MSO). TL does not only generalize *XPath*, XQuery and XSLT, but can also be simulated using MTT. The composition of MTT and their property of preserving recognizability for the calculation of their inverses are exploited to perform *inverse type inference*: they pre-compute in this way the pre-image of ill-formed output and perform type checking by simply testing whether the input type has some intersection with the pre-image.[2] Their system, as ours, is exact and does not approximate the computation but, in contrast to our method, there is a potential implementation problem (*i.e.*, an exponential blow-up) for the translation of MSO patterns into equivalent finite automata, on top on which most of their system is developed, even if MSO is not the only suitable pattern language that can be used with their system. Thus, our more specific approach, focused on a specific set of transformations, allows for a simpler (and more efficient) implementation.

Our approach complements work on XML schema evolution developed in the XML Schema context [GMS07, CGM11b], where validity preserving schema updates are identified and automatic adaptations identified, when possible. In case no automatic adaptation can be identified, the use of user-defined adaptation is proposed, but then a run-time (incremental) revalidation of all the

[1] The interested reader could refer to [BC09] (Section "Semantics"), where a translation of XQUF update expressions into a pending update list, made only of primitive operations, is provided, according to the W3C specification [Don09].

[2] Note that tree languages are closed under intersection and that the emptiness test is decidable for them.

adapted documents is needed. Similarly, in [GLQ11] a unifying framework for determining the effects of XML Schema evolution both on the validity of documents and on queries is proposed. The proposed system analyzes various scenarios in which forward/backward compatibility of schemas is broken. In [RS07], a related but different problem is addressed: how to exploit the knowledge that a given document is valid with respect to a schema S to (efficiently) assess its validity with respect to a different schema S'.

Another approach is proposed by the author of [ACHFR14]. An XML schema is modeled as a tree grammar, and each production is then encoded as a tree. Given a source S and a target schema T, a set of update operations U (*i.e.*, an edit-script) transforming S into T is automatically computed. Starting from the single operations of U, corresponding document update operations are derived. For any invalid document t, among the ones obtained in this way, the algorithm proposed in [ABS14] is employed for computing a correction. This algorithm, having exponential complexity, computes all the valid trees t' (with respect to a given grammar G playing the role of a schema), and having an edit-distance (from t) of at most a given threshold th. As already discussed in Section 2.1, despite document adaptations can be automatically generated by analyzing the syntactical variations of the schema, the intended semantics cannot be automatically captured. For this reason, the need for an efficient validation of user-generated document adaptations, as the one proposed in this chapter, is still present.

Finally, document update transformation is addressed in [BCG$^+$11], which investigates how to rewrite (document) updates specified on a view into (document) updates on the source documents, given the XML view definition.

2.3 Preliminaries

In this section we introduce the notations and definitions (mainly from [CDG$^+$07]) used in the remainder of the chapter.

Trees We refer to *terms* and *trees* as synonyms, as in [CDG$^+$07]. Given a string $s \in L$, with $L \subseteq \Sigma^*$, the set of its prefixes w.r.t. L is defined as $Pref_L(s) = \{t \mid s = tu \wedge t \in L \wedge u \in \Sigma^*\}$. Given a language $L \subseteq \Sigma^*$, we call *prefix language* the set of the prefixes of the elements of L: $Prefixes(L) = \bigcup_{s \in L} Pref_L(s)$. A language $L \subseteq \Sigma^*$ is said to be prefix-closed if $Prefixes(L) = L$, that is, if the language contains every possible prefix of every string belonging to the language itself.

A term is an element of a ranked alphabet, defined as the pair $\langle \Sigma, Arity \rangle$, where Σ is a finite and nonempty alphabet, and where $Arity \colon \Sigma \to \mathbb{N}$ is a function that associates a natural number, called *arity* of the symbol, with every element of Σ. The set of symbols with arity p is denoted as Σ_p. For the sake of conciseness, we employ a compact notation for terms, and we omit the set of

symbols it belongs to (*e.g.*, $f(,,)$ is a term contained in Σ_3). Σ_0 is called the set of *constants*. Let $\mathcal{X}$ be a set of *variables*, disjoint from Σ_0. The set $T(\Sigma, \mathcal{X})$ of the terms over Σ and $\mathcal{X}$ is defined as:

1. $\Sigma_0 \subseteq T(\Sigma, \mathcal{X})$,
2. $\mathcal{X} \subseteq T(\Sigma, \mathcal{X})$,
3. if $f \in \Sigma_p$, $p > 0$ and $t_1, \ldots, t_p \in T(\Sigma, \mathcal{X})$, then $f(t_1, \ldots, t_p) \in T(\Sigma, \mathcal{X})$.

If $\mathcal{X} = \emptyset$, we use $T(\Sigma)$ for $T(\Sigma, \mathcal{X})$ and its elements are called *ground terms*, that is, terms without variables. *Linear terms* are the elements of $T(\Sigma, \mathcal{X})$ in which each variable occurs at most once. An *unranked tree* t with labels belonging to a set of unranked symbols Σ is a map $t\colon \mathbb{N}^* \to \Sigma$ with a domain, denoted as $\mathcal{P}os(t)$, with the following properties:

1. $\mathcal{P}os(t)$ is a finite, nonempty and prefix-closed,
2. $\forall p \in \mathcal{P}os(t)\ .\ \exists k \geq 0\ .\ \forall i \in \{1, \ldots, k\}\ .\ p.i \in \mathcal{P}os(t)$.

In the remainder of the chapter, when no confusion arises, we refer to unranked trees simply as trees. $Root(t) = t(\epsilon)$ is called the *root* of tree t. The *subtree* $t|_p \in T(\Sigma, \mathcal{X})$ is the subtree at position p in a tree $t \in T(\Sigma, \mathcal{X})$ such that $\mathcal{P}os(t|_p) = \{j \mid p.j \in \mathcal{P}os(t)\}$ and $\forall q \in \mathcal{P}os(t|_p)\ .\ t|_p(q) = t(p.q)$. In what follows, we may refer to a subtree by means of (the position of) its root node.

Given a tree $t \in T(\Sigma, \mathcal{X})$, $Height$ function is inductively defined as follows:

- $Height(t) = 0$ if $t \in \mathcal{X}$,
- $Height(t) = 1$ if $t \in \Sigma_0$,
- $Height(t) = 1 + \underset{p \in \mathcal{P}os(t) \cap \mathbb{N}}{\arg\max}\ Height(t|_p)$.

The set of trees over Σ is denoted as $T(\Sigma)$.

An example of tree is $t = a(b(a, c(b)), c, a(a, c))$. Note that the same label can be used in different nodes which may have an arbitrary (but finite) number of children. An example of subtree is $t|_1 = b(a, c(b))$.

2.4 Hedge Automata and XML Schemas

Tree Automata (TA) are a natural generalization of finite-state automata, that is used to define languages over finite ranked trees (instead of finite words). TA can be used as a formal support for document validation [Mur97a, Mur97b]. In this setting, however, it is often more convenient to consider more general classes of automata, like Hedge and Sheaves Automata [ZL03], to manipulate both ranked and unranked trees. TA can model situations in which the same label present a variable number of children in a bounded range. This is achieved by associating a distinct rule with each possible value in the considered range. However, XML schemas allow an unbounded sequence of children to be associated with a given node. This requires on one hand the introduction of unranked trees to conveniently model XML documents, on the other a suitable formalism to synthetically express collections of such trees that obey to a finite set of constraints (called tree types). Hedge Automata (HA) are a suitable formal tool for reasoning on a representation of XML documents via unranked trees. HA are a generalization of TA because in TA, even if unranked symbols are supported, the horizontal languages are bounded sequences of states (representing set of trees). Therefore, TA do not support unbounded sequences of subtrees, as required in XML context.

Formally, given a tree $a(t_1, \ldots, t_n)$ where $n \geq 0$, the sequence $t_1, \ldots, t_n$ is called hedge. For $n = 0$ we have an empty sequence, represented by the symbol ϵ. The set of ground hedges over Σ is denoted as $H(\Sigma)$, while the set of hedges over symbols in Σ and variables in $\mathcal{X}$ is denoted as $H(\Sigma, \mathcal{X})$. In [BKMW01], hedges over Σ are inductively defined as follows:

- the empty sequence ϵ is a hedge,
- if g is a hedge and $a \in \Sigma$, then $a(g)$ is a hedge,
- if g and h are hedges, then gh is a hedge.

For instance, given a tree $t = a(b(a, c(b)), c, a(a, c))$, the corresponding hedges having as root nodes the children of $Root(t)$ are $b(a, c(b))$, c and $a(a, c)$. Hedge Automata are then formally defined in Definition 2.1 below.

Definition 2.1. A *Nondeterministic Finite Hedge Automaton* (NFHA) defined over Σ is a tuple $M = \langle \Sigma, Q, Q_f, \Delta \rangle$ where Σ is a finite and non empty alphabet, Q is a finite set of states, $Q_f \subseteq Q$ is the set of final states, also called accepting states, Δ is a finite set of transition rules of the form $a(\mathcal{HL}) \rightarrow q$, where $a \in \Sigma$, $q \in Q$ and $\mathcal{HL} \subseteq Q^*$ is a regular language over Q. $\triangle$

In the remainder of the chapter, when not explicitly stated, we refer to NFHA even if the more general term HA is adopted. Regular languages denoted as $\mathcal{HL}$ that appear in rules in Δ are said *horizontal languages*, and represented with Nondeterministic Finite Automata (NFA). We

represent regular languages by means of regular expressions with a standard notation. Specifically, given two regular expressions r and s, we denote their concatenation as rs, their union as $r|s$, and the unbounded repetition of r as r^*. The use of regular languages allows us to consider unranked trees. For instance, $a(q^*)$ matches a node a having an unbounded number of subtrees generated by state q. In Definition 2.2 we define HA computations.

Definition 2.2. A computation of M over a tree $t \in T(\Sigma)$ is a tree $M||t$, having the same domain as t. For every element $p \in \mathcal{P}os(M||t)$ such that $t(p) = a$ and $(M||t)(p) = q$, a rule $a(\mathcal{HL}) \to q$ in Δ must exist. If element p has n successors $p.1, \ldots, p.n$ such that $(M||t)(p.1) = q_1, \ldots, (M||t)(p.n) = q_n$, then $q_1 \cdots q_n \in \mathcal{HL}$. If $n = 0$ (that is, considering a leaf node) the empty string ϵ must belong to the language $\mathcal{HL}$ of the rule to be applied to the leaf node. $\triangle$

In Definition 2.3 we define, on top of the concept of HA computation, when a tree is accepted by an automaton, while in Definition 2.4 we define the language accepted by an automaton.

Definition 2.3. Given a HA $M = \langle \Sigma, Q, Q_f, \Delta \rangle$, a tree t is said to be *accepted* by M if a computation in which the root node has a label $q \in Q_f$ exists. $\triangle$

Definition 2.4. The *accepted language* for an automaton M, denoted as $L(M) \subseteq T(\Sigma)$, is the set of all the trees accepted by M. $\triangle$

In Definition 2.5 we formally define *tree types* on top of HA and computations.

Definition 2.5. Given a HA $M = \langle \Sigma, Q, Q_f, \Delta \rangle$, any state $q \in Q$ is a *(tree) type*. The language identified by q, denoted as $L(q)_M$, corresponds to $\{t \mid Root(M||t) = q\}$. $\triangle$

When HA M is clear from the context we use $L(q)$ in place of $L(q)_M$. When we say that HA M is a type, it is a shorthand for the tree type whose language corresponds to $L(M)$.

Example 2.1. Consider the HA $M = \langle \Sigma, Q, Q_f, \Delta \rangle$ where $\Sigma = \{0, 1, \boldsymbol{not}, \boldsymbol{and}, \boldsymbol{or}\}$, $Q = \{q_0, q_1\}$, $Q_f = \{q_1\}$ and $\Delta = \{\boldsymbol{not}(q_0) \to q_1, \boldsymbol{not}(q_1) \to q_0, 1(\epsilon) \to q_1, 0(\epsilon) \to q_0, \boldsymbol{and}(Q^* q_0 Q^*) \to q_0, \boldsymbol{and}(q_1 q_1^*) \to q_1, \boldsymbol{or}(Q^* q_1 Q^*) \to q_1, \boldsymbol{or}(q_0 q_0^*) \to q_0\}$. When used in a horizontal language, Q is simply a shorthand for $(q_0|q_1)$. Figure 2.1 shows a tree t representing a Boolean formula. Figure 2.2 shows the accepting computation of the automaton M (*i.e.*, $(M||t)(\epsilon) = q_1 \in Q_f$). Note that, despite logical **and** and **or** are binary operators, we exploit their associativity to consider them as unranked symbols. The corresponding TA differs from the HA only in the rules for these binary operators $\Delta = \{\ldots, \boldsymbol{and}(q_0, q_0) \to q_0, \boldsymbol{and}(q_0, q_1) \to q_0, \boldsymbol{and}(q_1, q_0) \to q_0, \boldsymbol{and}(q_1, q_1) \to q_1, \boldsymbol{or}(q_0, q_0) \to q_0, \boldsymbol{or}(q_0, q_1) \to q_1, \boldsymbol{or}(q_1, q_0) \to q_1, \boldsymbol{or}(q_1, q_1) \to q_1, \ldots\}$. $\Diamond$

A HA $M = \langle \Sigma, Q, Q_f, \Delta \rangle$ is said to be *normalized* if, for each $a \in \Sigma, q \in Q$, at most one rule $a(\mathcal{HL}) \to q \in \Delta$ exists. Since regular (word) languages are closed under union [CDG$^+$07], it

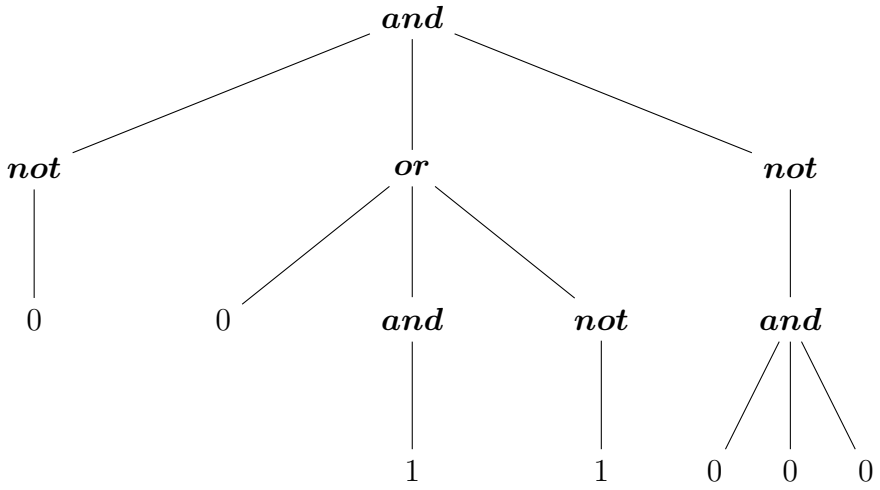


Figure 2.1: Tree t representing a true Boolean formula.

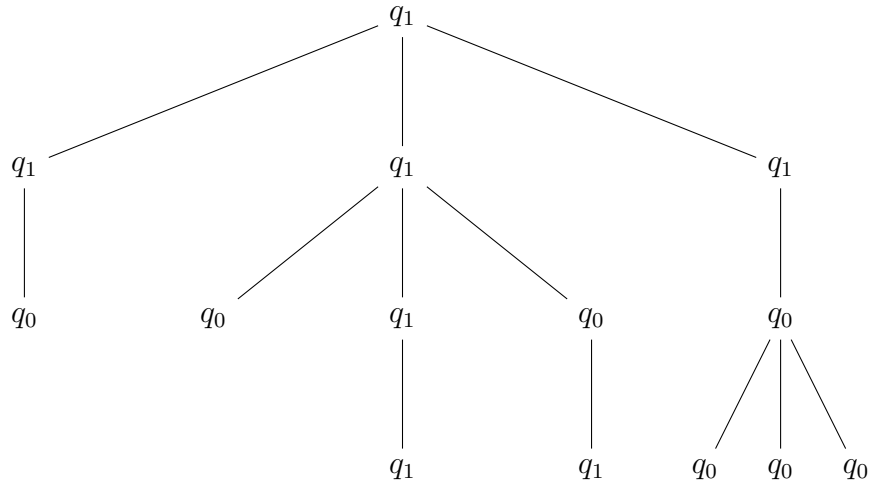


Figure 2.2: Accepting computation $M||t$ of the automaton M over tree t.

XQUF Primitive Update Operation	Update Rule
REN	$a(x) \rightarrow b(x)$
RPL	$a(x) \rightarrow p$
DEL	$a(x) \rightarrow ()$
INS_{first}	$a(x) \rightarrow a(px)$
INS_{last}	$a(x) \rightarrow a(xp)$
INS_{into}	$a(x) \rightarrow a(xpy)$
INS_{before}	$a(x) \rightarrow pa(x)$
INS_{after}	$a(x) \rightarrow a(x)p$

Table 2.1: XQUF primitives. a and b are XML tags, p is a state of a HA, and x, y are free variables that denote arbitrary sequences of states of the HA associated with the schema.

is always possible to define a normalized automaton starting from a non normalized HA. Every pair of rules $a(\mathcal{HL}_1) \rightarrow q$ and $a(\mathcal{HL}_2) \rightarrow q$ belonging to Δ is substituted by the equivalent rule $a(\mathcal{HL}_1 \cup \mathcal{HL}_2) \rightarrow q$.

Given two NFHA M_1 and M_2, the *inclusion test* consists in checking whether $L(M_1) \subseteq L(M_2)$. It can be reduced to the emptiness test for NFHA ($L(M_1) \subseteq L(M_2) \Leftrightarrow L(M_1) \cap (T(\Sigma) \setminus L(M_2)) = \emptyset$). Inclusion test is decidable, since complement, intersection, and emptiness of NFHA can be algorithmically executed [CDG$^+$07].

2.5 XQuery Update Facility as Parallel Rewriting

XQUF [Don09] is the standard, *W3C*-endorsed, update language for XML. Its expressions are converted into an intermediate format, called *Pending Update List* (PUL). Following [JR10], we consider here a formulation of primitive update operations of PUL via the set of rewriting rules defined in Table 2.1.

2.5.1 XQuery Update Facility

In Table 2.1 target node selection conditions are based on node labels only (and not on hierarchical relationships among the nodes). This limits the schema expressivity support to local schemas [MLMK05]. In Table 2.1, a and b are node labels, and p is a tree type. The supported update primitives allow for renaming an element (REN), replacing a subtree (RPL), deleting a subtree (DEL), inserting a subtree as a first, last, or an arbitrarily positioned child of an element ($INS_{first}, INS_{last}, INS_{into}$, respectively) and inserting a subtree before or after a given element ($INS_{before}, INS_{after}$, respectively). According to [Don09], the semantics (*i.e.*, the actual insertion position) of INS_{into} is implementation dependent. In real systems, this operation is often

not provided, or it is replaced by an INS_{first} or INS_{last} operation. For improving readability, in the rules we omit the character *comma*, used for separating subterms (*i.e.*, xy stands for x, y). We also refer to rule instances simply as rules.

As an example, consider the rule $REN\ a(x) \rightarrow b(x)$. Given a tree t, the rule must be applied to every target node (*i.e.*, every node having label a, given that x is a free variable that matches any sequence of subtrees of t). When the rule is applied to a node n with label a and children $t_1, \ldots, t_k$, the result of its application is the renaming of a into b, *i.e.*, the subterm $a(t_1, \ldots, t_k)$ is replaced by the subterm $b(t_1, \ldots, t_k)$. Consider now the rule INS_{first}, defined as $a(x) \rightarrow a(px)$, where p is a type (a state of a HA). Given a tree t, the rule must be applied to every node with label a. If the rule is applied to node n with label a and children $t_1, \ldots, t_k$, the result of its application is the insertion of a (nondeterministically chosen) term t of type p in the first position of the sequence of children of the target node, *i.e.*, the subtree $a(t_1, \ldots, t_k)$ is replaced by the subterm $a(t, t_1, \ldots, t_k)$. To capture the semantics of *XQUF* primitives that update, in parallel, every target node, we define a maximal parallel rewriting semantics, denoted via $\Rightarrow_r$ relation.

In order to formally define our rewriting system, we first need to introduce the general class of rules we adopt here and then we specify the semantics needed to model document adaptations.

2.5.2 Parameterized Hedge Rewriting System

Let $A = \langle \Sigma, Q, Q_f, \Delta \rangle$ be a HA called *parameterizing automaton*, whose states are used as types in the rules. A Parameterized Hedge Rewriting System (PHRS) [JR10] R/A is a set of hedge rewriting rules of the form $\mathcal{L} \rightarrow \mathcal{R}$, where $\mathcal{L} \in H(\Sigma, \mathcal{X})$, and $\mathcal{R} \in H(\Sigma \uplus Q, \mathcal{X})$. All the PHRS we consider in what follows have all the rules of R as instances of update rules of Table 2.1, and the previous definition forces type p appearing in such rules to be one of the states of the parameterizing HA (*i.e.*, states in Q). As in Table 2.1, we restrict our attention to linear rewriting rules (with a single occurrence of each variable in the left-hand and right-hand side). In [JR10, Tou12] the operational semantics is given via one-step rewriting (both the rule and the term to which a rule is applied are chosen in a nondeterministic way). Instead, XML document updates have a global effect. For instance, when renaming a label in an XML document, all the nodes having that label must be renamed. Such an update may be expressible through maximal steps of sequential applications of the REN rewriting rule. However, maximal sequential rewriting is not applicable to insertion rules like $INS_{first} = a(x) \rightarrow a(px)$: sequential applications of INS_{first} may select a single target node more than once, thus yielding incorrect results, as shown in Example 2.2.

Example 2.2. Consider the terms $t = a(a(b, c), b)$ and $t' = d(e)$, and the rule r defined as $a(x) \rightarrow a(t'x)$ (insertion of t' into t as first child of all the nodes labelled by a). The intended semantics of INS_{first} applies r to all the matching occurrences of $a(x)$ in t, therefore yielding the new term $a(d(e), a(d(e), b, c), b)$. On the other hand, via the one-step rewriting, we may

generate trees like $t_1 = a(a(d(e), d(e), b, c), b)$, and $t_2 = a(d(e), d(e), a(b, c), b)$ that are not correct transformations of t. $\Diamond$

2.5.3 Parallel Rewriting

In order to capture the behavior of update rules we introduce a new parallel rewriting semantics for PHRS, based on tree contexts and substitutions [CDG$^+$07]. To this aim, we first define this basic notations, then the intuition underlying the parallel rewriting is given, finally we provide its formal definition.

Basic Definitions A substitution is a map $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ that substitutes x_i with t_i, where $i \in [1 \ldots n]$. Let $t \in T(\Sigma, \{x\}_n)$ be a term, and $\sigma = \{x \leftarrow t'\}$ be a substitution. We denote as σt the application of σ to t, that substitutes each occurrence of variable x in t, with term t'. Note that, from the definition of terms with variable given in Section 2.3, variable nodes cannot have children. A context $C \in T(\Sigma, \mathcal{X})$ is a linear term, that is, a term where each variable can occur at most once. Here we consider $\mathcal{X} = \{\bar{y}\}$. Given a context C, we define the substitution of the variable $\bar{y}$ of C with tree $t \in T(\Sigma)$, denoted as $C[t] \in T(\Sigma)$, as the tree obtained from the substitution of $\bar{y}$ with t, that is: $C[t] = C[\bar{y} \leftarrow t]$. Let t be a tree, and $r = \mathcal{L} \rightarrow \mathcal{R}$ be an update rule of Table 2.1. We denote as $symbol(r)$ the symbol appearing in $\mathcal{L}$. As for any rewriting rule, subtrees matching the left-hand side of r are replaced by the right-hand side of the same rule. These subtrees are called target subtrees, and we denote as $Target(t, r)$ the list of positions of their root nodes, called target positions. Their root nodes have $symbol(r)$ symbol as label, and are called target nodes.

Intuition A parallel rewriting step of rule r on a tree t transforms it into a new tree t' and implies the rewriting of each target node, according to rule r. Target positions are expressed w.r.t. tree t, and given that intermediate rewritings may insert nodes, they are not guaranteed to be associated to the same node as in t. It is clear from the update rules, that only successors of a target node may "change position" in the updated tree. Therefore, in order to address this problem, we order the elements of $Target(t, r)$ list, according to the lexicographic ordering. Such an order intuitively corresponds to that in which target positions may be encountered during a bottom-up visit of tree t. We define the parallel semantics in a more general way, such that if the considered rule r has a tree type s in its right-hand side, we may instantiate it, for each target position, with a different tree of $L(s)$, similarly to what is considered for the standard *tree concatenation* operation [Tho90].

We now introduce the basic ideas underlying the parallel rewriting, by means of an example. Let $r = b(x) \rightarrow b(sx)$ be a rule, such that $L(s) = \{c_1, c_2\}$, let $t_i = a(b(\underline{b(c_1)}))$ be the tree considered at an intermediate rewriting step, and $p_i = 1$ be the target position to be processed.

The underlined subtree has been already rewritten. We may explain the effect of this single rewriting step by decomposing t_i into different components. Let $C_i = a(\bar{y})$ be the context obtained from t_i by replacing the subtree rooted at p_i, with variable $\bar{y}$. For computing the tree $\mathcal{R}$ to substitute in context C_i, we need to instantiate type s and variable x in the right-hand side of r. The first instantiation is a nondeterministic choice of a tree in $L(s)$, suppose c_2. The second instantiation requires to replace x with the forest represented by the children subtrees of the target node at position p_i, forest that is in this way "copied" in the new tree without being affected. The effect of applying this intermediate rewrite step is then equal to $C_i[\mathcal{R}] = a(b(c_2, b(c_1)))$. Example 2.3 shows a complete example of parallel rewriting.

Example 2.3. Consider term $t = b(c, d(c(a), a))$, and the rule $r = c(x) \rightarrow c(x)p$, where $t_2 = a(b)$, $t_1 = a(c(a), c(a)) \in L(p)$. The set of positions in t is defined as $\mathcal{P}os(t) = \{\epsilon, 1, 2, 2.1, 2.2, 2.1.1\}$. The rule r matches nodes of t at positions 1 and 2.1. We start from position 2.1, and we compute the context C_2 defined by replacing the subtree at position 2.1 with a fresh variable $\bar{y}$. We obtain the term $b(c, d(\bar{y}, a))$. The substitution $\sigma_2 = \{x \leftarrow (a)\}$ is the result of matching $c(x)$ with $c(a)$. We can now rewrite the context $C_2[\bar{y}]$ as $C_2[\mathcal{R}_2\sigma_2]$, where $\mathcal{R}_2$ is obtained by instantiating p with term t_2. Clearly, the free variable $\bar{y}$ is not strictly necessary since we can directly insert the new term in the current position. However, we use it in our examples to isolate the position in which the rewriting step takes place. This gives us the intermediate tree $b(c, d(c(a), \underline{a(b)}, a))$ (the new subtree is underlined). We now consider position 1, we extract the context $\overline{C_1} = b(\bar{y}, d(c(a), a(b), a))$ and consider the substitution $\sigma_1 = \{x \leftarrow \epsilon\}$ between $c(x)$ and c. We apply the rewrite step by substituting $\bar{y}$ with $\mathcal{R}_1\sigma_1$, and we obtain the term $C_1[\mathcal{R}_1\sigma_1] = b(c, \underline{a(c(a), c(a))}, d(c(a), a(b), a))$ (the inserted subtree is underlined) corresponding to the result of the parallel rewriting step. $\Diamond$

Formal Definition As a preliminary step for the definition of the parallel rewriting semantics, we formally define its basic components.

In Definition 2.6 we first introduce $<_{lex}$ as the lexicographic ordering for strings in $\mathbb{N}^*$. This ordering is then used to process target positions. It exploits the fact that our rewriting rules, when applied to a position p, do not affect any node position that precedes p w.r.t. $<_{lex}$. In practice, the ordering allows to apply rewritings on target nodes in a way that does not change the position of target nodes that still need to be rewritten. These positions are expressed as strings over $\mathbb{N}^*$, where symbol "." is used as character separator. Given a string s, we denote as $s[i]$ the *i-th* character.

Definition 2.6. Given two strings x, y in $\mathbb{N}^*$, we say that x (*lexicographically*) *precedes* y, denoted as $x <_{lex} y$, iff:

- $|x| < |y|$, or
- $\exists i \in [1 \ldots |y|] \ . \ \forall j < i \ . \ x[j] = y[j] \wedge x[i] < y[i]$. $\triangle$

Notice that, over the single characters of a position (that are elements of $\mathbb{N}$), the ordering simply corresponds to the standard relation $<$ over natural numbers. In Definition 2.7 the sequence of target positions matching the guard (*i.e.*, the left-hand side) of a rule, denoted as $Target$, is defined.

Definition 2.7. Given a tree $t \in T(\Sigma)$, a PHRS R/A with rules in $XQUF$, and a rule $r = \mathcal{L} \to \mathcal{R} \in R$, we denote the *ordered list of target nodes of* t *w.r.t.* r as $Target(t, r) = (p_1, \ldots, p_n)$ such that, for all $i \in [1 \ldots n-1]$, $p_{i+1} <_{lex} p_i$ and $t(p_i) = symbol(r)$, with $n \geq 0$. △

In Definition 2.8, given a tree $t \in T(\Sigma)$, we define the forest (ordered sequence) of the subtrees rooted at its children.

Definition 2.8. Given a tree $t = a(t_1, \ldots, t_n)$, we define the *forest obtained by deleting its root* as $subt(t) = (t_1, \ldots, t_n)$. When an optional position $i \in [0 \ldots n]$ is provided, the function returns two (possibly empty) subforests, splitted at position i, and denoted as $subt(t, i) = ((t_1, \ldots, t_i), (t_{i+1}, \ldots, t_n))$. △

Finally, given a tree t, a position $p \in \mathcal{P}os(t)$, and a (list of) term(s) l, we define the term $rep(t, p, l)$ as the term t' obtained by replacing the (list of) subterm(s) at position p in t, with l. More formally, $t'|_p = l$ and $t'|_q = t|_q$, if p is not a prefix of q.

We are now ready to define, in Definition 2.9, the parallel rewriting relation $\Rightarrow_r$, associated with a rule $r = \mathcal{L} \to \mathcal{R}$. In the definition we consider instances $\mathcal{R}'$ of $\mathcal{R}$ obtained by instantiating types with nondeterministically chosen terms from the corresponding type language. Clearly, there are infinitely many possible instances. Furthermore, distinct instances can be used in different rewritings within a single parallel step (as in Example 2.3). As previously noticed, this general behavior is similar to that of the standard *tree concatenation* operation [Tho90].

Definition 2.9. For $t, t' \in T(\Sigma)$, let $r = \mathcal{L} \to \mathcal{R}$ be an $XQUF \setminus \{INS_{into}\}$ rule such that x is the free variable in r and $Target(t, r) = (p_1, \ldots, p_n)$ for $n > 0$.
Then, t and t' are in the *parallel rewriting relation*, denoted as $t \Rightarrow_r t'$, iff $t' \in S_t^{n,n}$, where $S_t^{n,n}$ is the set of trees inductively defined as follows:

$$S_t^{0,n} = \{t\}$$

$$S_t^{i,n} = \left\{ t' \;\middle|\; \begin{array}{l} t' = rep(t, p_i, \mathcal{R}'\sigma_i), \\ \mathcal{R}' \; is \; an \; instance \; of \; \mathcal{R}, \\ \sigma_i = \{x \leftarrow subt(t''|_{p_i})\}, \\ t'' \in S_t^{i-1,n} \end{array} \right\} \; for \; i \in [1 \ldots n]$$

△

The substitution σ_1 (for the sole variable x occurring in r) is the result of matching the left-hand side of the rule with the subterm of t at position p_1 (the first processed target position). For

$i \in [1 \dots n]$, the substitution σ_i is recursively defined on top of $terms$ in $S_t^{i-1,n}$, to preserve the already applied rewritings.

Let p be the type occurring in the right-hand side $\mathcal{R}$ of a rule r. In this case, $S_t^{n,n}$ is the set of trees obtained through a bottom-up traversal of t, imposed by the ordering on the target positions. Intermediate rewritings preserve the effect of the previous ones, occurred at larger positions (*i.e.*, closer to the tree leaves). As already discussed, (possibly) different trees of type p may be used for instantiating $\mathcal{R}$, at each step. For this reason, instead of a single successor for a tree w.r.t. the *i-th* rewriting step, we define a set of successor trees, denoted as $S_t^{i,n}$.

Clearly, this generalizes the simpler case in which a single tree t is used for all the rewritings. This behavior may be simulated by using a type p_t, whose language coincides with the singleton of t. The generalized semantics can be viewed as an abstraction of conditional parallel rewriting that, depending on conditions defined on target position, may insert different trees, belonging to the same type, in each document transformation.

The definition for the INS_{into} operation differs in that, even for the simple case (*i.e.*, when types whose language is a singleton are employed), we may still have several possible successor terms, induced by the nondeterminism in choosing the insertion position between those of the children of the target node. Indeed, the definition of $S_t^{i,n}$ is changed as follows and restated in Definition 2.10.

Definition 2.10. Let x, y be the two free variables of INS_{into} rule $r = \mathcal{L} \rightarrow \mathcal{R}$:

$$S_t^{0,n} = \{t\}$$

$$S_t^{i,n} = \left\{ t' \;\middle|\; \begin{array}{l} t' = rep(t, p_i, \mathcal{R}'\sigma_i), \\ \mathcal{R}' \; is \; an \; instance \; of \; \mathcal{R}, \\ subt(t''|_{p_i}, j) = l_1, l_2, \; for \; j \in [0 \dots n+1], \\ \sigma_i = \{x \leftarrow l_1, y \leftarrow l_2\}, \\ t'' \in S_t^{i-1,n} \end{array} \right\} for \; i \in [1 \dots n]$$

△

The definition can be extended, in a natural way, to a set of rules R of a PHRS (based on *XQUF* primitives), yielding the new relation $\Rightarrow_R$. The reflexive-transitive closure of $\Rightarrow_R$ relation is denoted as $\Rightarrow_R^*$.

Finally, given a PHRS R/A based on *XQUF* rules, a sequence $\upsilon = (u_1, \dots, u_n)$ of update operations (that is, a document adaptation) such that $u_i \in R$ for $i \in [1 \dots n]$ we define $Post_\upsilon(S)$, where $S \subseteq T(\Sigma, \mathcal{X})$, as the language obtained by a single application of each rule of υ to each element of S, under parallel rewriting semantics. When υ is clear from the context, the shorthand $Post(S)$ is employed.

The following examples illustrate the behavior of the parallel rewriting semantics on different update rules. For clarity, we present the rewritings in terms of contexts (with variable $\bar{y}$) and substitutions. We denote the left-hand (resp. right-hand) side of rule r as $\mathcal{L}$ (resp. $\mathcal{R}$). We also denote with t,t' two trees such that $t \Rightarrow_r t'$. Furthermore, we denote as $t^{i,n}$, terms in $S_t^{i,n}$.

Example 2.4. Consider the RPL rule $r = c(x) \to p$, with t, p, $\mathcal{P}os(t)$ and $Target(t, r)$ as in Example 2.3.

- $C_1 = b(c, d(\bar{y}, a)), \sigma_1 = \{x \leftarrow (a)\}$,
- $t^{1,2} = C_1[\mathcal{R}\sigma_1] = b(c, d(a(b), a))$,
- $C_2 = b(\bar{y}, d(a(b), a)), \sigma_2 = \{x \leftarrow (\epsilon)\}$,
- $t^{2,2} = C_2[\mathcal{R}\sigma_2] = b(a(b), d(a(b), a)) = t'$. $\Diamond$

Example 2.5. Consider now the DEL rule $r = c(x) \to ()$. Let t, p, $\mathcal{P}os(t)$ and $Target(t, r)$ be as in Example 2.3

- $C_1 = b(c, d(\bar{y}, a)), \sigma_1 = \{x \leftarrow (a)\}$,
- $t^{1,2} = C_1[\mathcal{R}\sigma_1] = b(c, d(a))$,
- $C_2 = b(\bar{y}, d(a)), \sigma_2 = \{x \leftarrow (\epsilon)\}$,
- $t^{2,2} = C_2[\mathcal{R}\sigma_2] = C_2[()] = b(d(a)) = t'$. $\Diamond$

Example 2.6. Consider now the INS_{first} rule $r = a(x) \to a(px)$, with $t = a(b(c, d(a)), d(a(c), c))$, $p = c(d, b)$, $\mathcal{P}os(t) = \{\epsilon, 1, 2, 1.1, 1.2, 2.1, 2.2, 1.2.1, 2.1.1\}$, $Target(t, r) = (1.2.1, 2.1, \epsilon)$:

- $C_1 = a(b(c, d(\bar{y})), d(a(c), c)), \sigma_1 = \{x \leftarrow (\epsilon)\}$,
- $t^{1,3} = C_1[\mathcal{R}\sigma_1] = a(b(c, d(a(c(b, d)))), d(a(c), c))$,
- $C_2 = a(b(c, d(a(c(d, b)))), d(\bar{y}, c)), \sigma_2 = \{x \leftarrow (c)\}$,
- $t^{2,3} = C_2[\mathcal{R}\sigma_2] = a(b(c, d(a(c(d, b)))), d(a(c(d, b), c), c))$,
- $C_3 = \bar{y}, \sigma_3 = \{x \leftarrow (t_1, t_2)\}$, where $t_1 = b(c, d(a(c(d, b))))$ and $t_2 = d(a(c(d, b), c), c)$,
- $t^{3,3} = C_3[\mathcal{R}\sigma_3] = a(c(d, b), b(c, d(a(c(d, b)))), d(a(c(d, b), c), c)) = t'$. $\Diamond$

Example 2.7. Finally, consider the REN rule $r = c(x) \to c'(x)$, where $t = c(c(c(a)))$, $\mathcal{P}os(t) = \{\epsilon, 1, 1.1, 1.1.1\}$, $Target(t, r) = (1.1, 1, \epsilon)$:

- $C_1 = c(c(\bar{y})), \sigma_1 = \{x \leftarrow (a)\}$,
- $t^{1,3} = C_1[\mathcal{R}\sigma_1] = c(c(c'(a)))$,
- $C_2 = c(\bar{y}), \sigma_2 = \{x \leftarrow c'(a)\}$,
- $t^{2,3} = C_2[\mathcal{R}\sigma_2] = c(c'(c'(a)))$,
- $C_3 = \bar{y}, \sigma_3 = \{x \leftarrow c'(c'(a))\}$,
- $t^{3,3} = C_3[\mathcal{R}\sigma_3] = c'(c'(c'(a))) = t'$. $\Diamond$

2.6 Hedge Automata-based Static Analysis

In this section we describe the symbolic algorithm underlying the HASA module. Given a set of trees L, and an update rule r, we recall that $Post_r(L)$ denotes the set of updated trees $\{t' \mid t \Rightarrow_r t',\ t \in L\}$. As mentioned in the introduction, our goal is to effectively compute the result of a document adaptation on each tree that is accepted by a given HA A. We thus define a HA transformation from A to a new HA A' such that $L(A') = Post_r(L(A))$. In order to define such a transformation we need to operate both on (vertical) rules of A and on horizontal languages of rules of A. INS_{into} rule needs a special treatment. We anticipate that the nondeterminism in the choice of the insertion position may introduce the need of considering several successors w.r.t. $Post$ computation, for the same instance rule. In practical implementations this is avoided because the semantics of INS_{into} rule is always resolved in favour of some fixed insertion position. In what follows we describe the construction, we give the correctness proof of the algorithm w.r.t. our parallel semantics, and we discuss its temporal complexity.

2.6.1 Computing Post

Let $A = \langle \Sigma_o, P, P_f, \Theta \rangle$ be a HA, let R/A be a PHRS, let $A_L = \langle \Sigma_L, Q_L, Q_L^f, \Delta_L \rangle$ be a HA that describes a collection of documents, such that $P \cap Q_L = \emptyset$, $L = L(A_L)$. Let also υ be a document adaptation based on R/A. Taken this elements in input, our goal is to compute a HA $A' = \langle \Sigma_o \cup \Sigma_L, P \cup Q_L, Q_L^f, \Delta' \rangle$ such that $L(A') = Post_\upsilon(L)$. In particular, our method computes the set of transition relations Δ', from the elements of Δ_L, with rule type dependent modification patterns.

Without loss of generality, we consider A and A_L to be normalized automata (see Section 2.4). For each $a \in \Sigma$, $q \in Q_L$, we denote with $L_{a,q}$ the horizontal language of the unique rule $a(L_{a,q}) \rightarrow q \in \Delta_L$, accepted by the NFA $B_{a,q} = \langle Q_L, S_{a,q}, i_{a,q}, \{f_{a,q}\}, \Gamma_{a,q} \rangle$. A horizontal transition from state s to state s' that consumes a character c, is denoted as (s, c, s'). For brevity, we also denote the alphabet $\Sigma_o \cup \Sigma_L$ of HA A', as Σ.

As a preliminary operation, we need to expand the alphabet of each NFA, from Q_L to $P \cup Q_L$ (in order to support *XQUF* rules presenting a tree type in their right-hand side).

Revisiting Normalization

Before defining the transformations that define $Post$, we need to introduce a stronger form of normalization. Suppose that our document adaptation presents an operation $r = a(x) \rightarrow \mathcal{R}$, having type INS_{before}, INS_{after}, RPL, or DEL, and that state $q \in Q_L$ appearing in $\mathcal{R}$ is shared with symbols of Σ other than a (*i.e.*, we have one rule $a(L_a) \rightarrow q \in \Delta_L$, and at least one rule $b(L_b) \rightarrow q \in \Delta_L$ such that $a \neq b$). In this scenario, applying the changes imposed on Δ_L by the modification patterns triggered by rule r would not be restricted to symbol a only, but would also affect all the other symbols associated with state q. In order to be able to restrict the effects of our modification patterns, we have to ensure that each rule in Δ_L has a distinct target state. This transformation (called *strong normalization*) is defined as follows. For each rule $a(...) \rightarrow q \in \Delta_L$:

1. we create a fresh state $q_a^{fresh} \notin P \cup Q_L$, and we add it to Q_L,
2. for each rule $b(L_{b,q'}) \rightarrow q' \in \Delta_L$ such that q appears as a symbol in $L_{b,q'}$, we replace each occurrence of q with $(q|q_a^{fresh})$.

It is evident that the transformed automaton is equivalent to the input one. In Example 2.8 we present an example of such preliminary transformation.

Example 2.8. Consider the HA M of Example 2.1. Suppose we have a rule $\boldsymbol{not}(x) \rightarrow \boldsymbol{not}(px)$ in our document adaptation. Our transformation is applied to rules $\boldsymbol{not}(q_0) \rightarrow q_1$, and $q_1 \rightarrow q_0$. We expand the set of states of M, by adding the fresh states $q_{1_{not}}^{fresh}$ and $q_{0_{not}}^{fresh}$, and we replace rule $\boldsymbol{not}(q_0) \rightarrow q_1$ and $\boldsymbol{not}(q_1) \rightarrow q_0$ by rule $\boldsymbol{not}(q_0) \rightarrow q_{1_{not}}^{fresh}$ and $\boldsymbol{not}(q_1) \rightarrow q_{0_{not}}^{fresh}$, respectively. We denote as Q' this expanded set of states. We recall that in regular expressions, Q and Q' stands for the union of all their states. In what follows, we detail the transformations of the transition rules:

- $\boldsymbol{not}(q_0) \rightarrow q_{1_{not}}^{fresh}$ is replaced by $\boldsymbol{not}((q_0|q_{0_{not}}^{fresh})) \rightarrow q_{1_{not}}^{fresh}$,
- $\boldsymbol{and}(Q^*q_0Q^*) \rightarrow q_0$ is replaced by $\boldsymbol{and}(Q'^*(q_0|q_{0_{not}}^{fresh})Q'^*) \rightarrow q_0$,
- $\boldsymbol{or}(q_0q_0^*) \rightarrow q_0$ is replaced by $\boldsymbol{or}(q_0|q_{0_{not}}^{fresh})(q_0|q_{0_{not}}^{fresh})^*) \rightarrow q_0$,
- $\boldsymbol{not}(q_1) \rightarrow q_{0_{not}}^{fresh}$ is replaced by $\boldsymbol{not}((q_1|q_{1_{not}}^{fresh})) \rightarrow q_{0_{not}}^{fresh}$,
- $\boldsymbol{and}(q_1q_1^*) \rightarrow q_1$ is replaced by $\boldsymbol{and}(q_1|q_{1_{not}}^{fresh})(q_1|q_{1_{not}}^{fresh})^* \rightarrow q_1$,

- $\boldsymbol{or}(Q^*q_1Q^*) \rightarrow q_1$ is replaced by $\boldsymbol{or}(Q'^*(q_1|q_{1_{not}}^{fresh})Q'^*) \rightarrow q_1$. $\Diamond$

With this preliminary transformation in place, we can define the modification patterns for a rule $r \in R/A$, modifications depending on the type of r.

Renaming: REN

Let $r = a(x) \rightarrow b(x)$, where $a, b \in \Sigma$, for each $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$. If $L(B_{b,q}) = \emptyset$, $B_{b,q}$ simply coincides with $B_{a,q}$. Otherwise, if $L(B_{b,q}) \neq \emptyset$, $B_{b,q}$ is defined as the automaton that recognizes the union of $L(B_{a,q})$ and $L(B_{b,q})$ (*i.e.*, $B_{b,q} = \langle Q_L, S_{a,q} \uplus S_{b,q} \uplus \{i_{ab,q}\}, i_{ab,q}, \{f_{a,q}\} \uplus \{f_{b,q}\}, \Gamma_{a,q} \uplus \Gamma_{b,q} \uplus \{(i_{ab,q}, \epsilon, i_{a,q}), (i_{ab,q}, \epsilon, i_{b,q})\}\rangle$). As a final step, in both cases, each rule of the form $a(L_{a,q}) \rightarrow q$ in Δ_L, where $q \in Q_L$, is transformed into $b(L_{b,q}) \rightarrow q$. Intuitively, these changes allow any subtree $b(x)$ to be evaluated, by A', in state q, when the corresponding subtree $a(x)$ is evaluated by A_L in q. In the first case, the sequence of states x is accepted by $B_{b,q}$, the precondition for applying rule $b(L_{b,q}) \rightarrow q$. In the latter, analogously, x is accepted by $B_{a,q}$ and $a(L_{a,q}) \rightarrow q$ is applied.

Insert first: INS_{first}

Let $r = a(x) \rightarrow a(px)$. The modifications occur to each automaton $B_{a,q}$ such that $q \in Q_L$ and $L_{a,q} \neq \emptyset$. A fresh state $q_{a,q}^{fresh}$, such that $q_{a,q}^{fresh} \notin S_{a,q}$, is created. The fresh state is added to $S_{a,q}$ and it is now an initial state. If $\Gamma_{a,q} = \emptyset$ holds, $(q_{a,q}^{fresh}, p, f_{a,q})$ transition is added to $\Gamma_{a,q}$. Otherwise, for each transition of the form $(i_{a,q}, y, q_y) \in \Gamma_{a,q}$ such that $i_{a,q}$ is an initial state, $y \in P \cup Q_L$, $q_y \in S_{a,q}$, a transition of the form $(q_{a,q}^{fresh}, p, i_{a,q})$ is added to $\Gamma_{a,q}$.

Figure 2.3: The changes to the horizontal automaton, due to rule INS_{first}, are depicted as grey texts and dotted lines.

Insert last: INS_{last}

Let $r = a(x) \rightarrow a(xp)$. The modifications occur to each automaton $B_{a,q}$ such that $q \in Q_L$ and $L_{a,q} \neq \emptyset$. A fresh state $q_{a,q}^{fresh}$, such that $q_{a,q}^{fresh} \notin S_{a,q}$, is created. The fresh state is added to $S_{a,q}$ and it is now a final state. If $\Gamma_{a,q} = \emptyset$ holds, the transition $(i_{a,q}, p, q_{a,q}^{fresh})$ is added to $\Gamma_{a,q}$. Otherwise, for each rule of the form $(q_y, y, f_{a,q}) \in \Gamma_{a,q}$ such that $y \in P \cup Q_L$, $q_y \in S_{a,q}$, a transition of the form $(f_{a,q}, p, q_{a,q}^{fresh})$ is added to $\Gamma_{a,q}$.

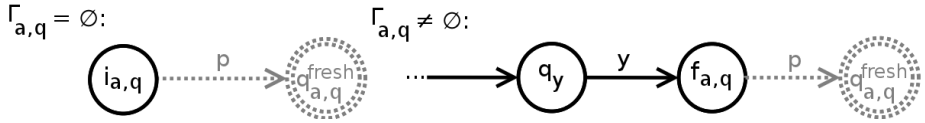


Figure 2.4: The changes to the horizontal automaton, due to rule INS_{last}, are depicted as grey texts and dotted lines.

Insert before: INS_{before}

Let $r = a(x) \rightarrow pa(x)$. The modifications occur to each horizontal language having at least an element in which a state $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$ occurs. For each $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$, a fresh state $q_{a,q}^{fresh}$ is created such that $q_{a,q}^{fresh} \notin S_{b,z}$, for each $b \in \Sigma$ and $z \in Q_L$. Then, $q_{a,q}^{fresh}$ is added to $S_{b,z}$ if at least one transition of the form $(s, q, s') \in \Gamma_{b,z}$ exists, with $s, s' \in S_{b,z}$. Such transition is replaced by $(s, p, q_{a,q}^{fresh})$, and the corresponding transition $(q_{a,q}^{fresh}$, $q, s')$ is added to $\Gamma_{b,z}$.

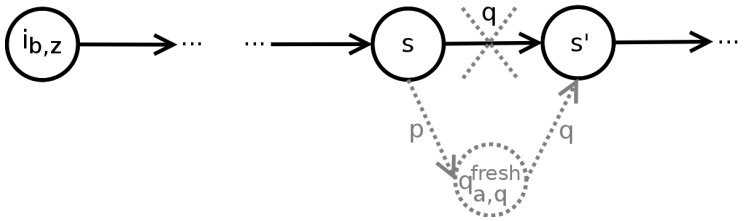


Figure 2.5: The changes to the horizontal automaton, due to rule INS_{before}, are depicted as grey texts and dotted lines.

Insert after: INS_{after}

Let $r = a(x) \rightarrow a(x)p$. The modifications occur to each horizontal language having at least an element in which a state $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$ occurs. For each $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$, a fresh state $q_{a,q}^{fresh}$ is created such that $q_{a,q}^{fresh} \notin S_{b,z}$, for each $b \in \Sigma$ and $z \in Q_L$. Then $q_{a,q}^{fresh}$ is added to $S_{b,z}$ if at least one transition of the form $(s, q, s') \in \Gamma_{b,z}$ exists, with $s, s' \in S_{b,z}$. Such transition is replaced by $(s, q, q_{a,q}^{fresh})$, and the corresponding transition $(q_{a,q}^{fresh}$, $p, s')$ is added to $\Gamma_{b,z}$.

Replace: RPL

Let $r = a(x) \rightarrow p$. The modifications occur to each horizontal language having at least an element in which a state $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$ occurs. Each transition of the form $(s, q$, $s') \in \Gamma_{b,z}$, with $b \in \Sigma$ and $z \in Q_L$, is replaced by (s, p, s').

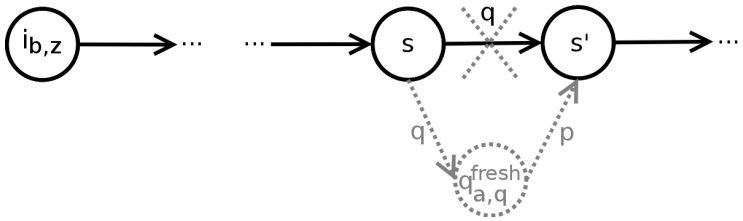


Figure 2.6: The changes to the horizontal automaton, due to rule INS_{after}, are depicted as grey texts and dotted lines.

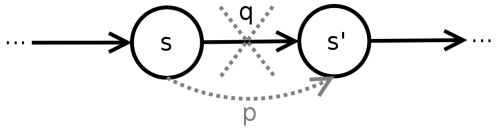


Figure 2.7: The changes to the horizontal automaton, due to rule RPL, are depicted as grey texts and dotted lines.

Delete: DEL

Let $r = a(x) \rightarrow ()$. The modifications occur to each horizontal language having at least an element in which a state $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$ occurs. Each transition of the form $(s, q, s') \in \Gamma_{b,z}$, with $b \in \Sigma$ and $z \in Q_L$, is replaced by (s, ϵ, s').

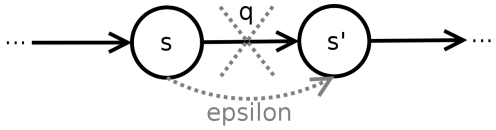


Figure 2.8: The changes to the horizontal automaton, due to rule DEL, are depicted as grey texts and dotted lines.

Insert into: INS_{into}

The simulation of the INS_{into} rule requires some care. The rule inserts a subtree in a nondeterministically chosen position between the children of a given node. Since the position is not known in advance, we can only guess a state s of the horizontal automata and replace its outgoing transitions with transitions passing through a fresh state. However, we may need to consider an automaton for every such state s. We describe next the $Post$ construction for a given choice of s. Let $r = a(x) \rightarrow a(xpy)$. The modifications occur to each automaton $B_{a,q}$ such that $q \in Q_L$ and $L_{a,q} \neq \emptyset$. A fresh state $q_{a,q}^{fresh}$, such that $q_{a,q}^{fresh} \notin S_{a,q}$, is created. The fresh state is added to $S_{a,q}$. At this point, for each state $s \in S_{a,q}$ reachable from $i_{a,q}$ through the transitions in $\Gamma_{a,q}$, each

transition of the form (s, j, s') is replaced by $(s, j, q_{a,q}^{fresh})$, where $j \in P \cup Q_L$ and $s' \in S_{a,q}$, and transitions of the form $(q_{a,q}^{fresh}, p, s')$ are added to $\Gamma_{a,q}$.

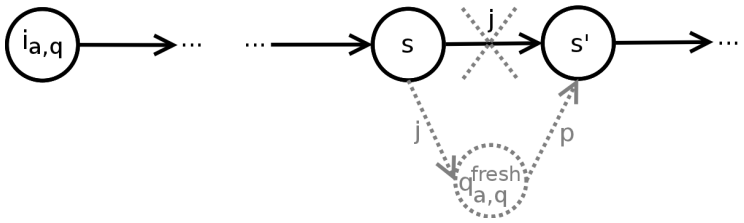


Figure 2.9: The changes to the horizontal automaton, due to rule INS_{into}, are depicted as grey texts and dotted lines.

The need of guessing the right position in the horizontal automata in which inserting a fresh state generates several possible HA for each occurrence of INS_{into} in the document adaptation. However, in real implementations, this operation often reduces either to INS_{first} or INS_{last}. Thus, in practical cases, this avoids the need of introducing a search procedure in our HASA module.

Finally, Δ' is defined as $\Theta \cup \{a(B_{a,q}) \to q \mid a \in \Sigma, q \in Q_L, L(B_{a,q}) \neq \emptyset)\}$. The transitions of Θ ensures that A' is able to evaluate any subtree belonging to L, the other transitions are used by A' for the evaluation of the elements of $L(A)$, modified according to the given document adaptation. The test $L(B_{a,q}) \neq \emptyset$ excludes useless transitions.

INS_{before}, INS_{after}, and DEL operations may invalidate a valid XML document when applied to the root node. Therefore, we do no allow the document adaptations that presents rules of the form $a(x) \to pa(x)$, $a(x) \to a(x)p$, or $a(x) \to ()$, when to any tree $t \in T(\Sigma)$ such that $t(\epsilon) = a$.

In the following we provide an example of application of the HASA module.

Example 2.9. Suppose we have two HA $A_L = \langle \Sigma_L, Q_L, Q_L^f, \Delta_L \rangle$ and $A = \langle \Sigma, P, P_f, \Theta \rangle$, defined as follows:

- $\Sigma_L = \{a, b, c\}$ and $\Sigma = \{a, b, d\}$,
- $Q_L = \{q_{a1}, q_{a2}, q_b, q_c\}$ and $P = \{g_a, g_b, g_d\}$,
- $Q_L^f = \{q_{a1}, q_{a2}\}$ and $P_f = \{g_a\}$,
- $\Delta_L = \{a({q_b}^*) \to q_{a2}, a({q_b}^* q_c) \to q_{a1}, b(\epsilon) \to q_b, c(\epsilon) \to q_c\}$,
- $\Theta = \{a({g_b}^+) \to g_a, b({g_b}^+ | g_d) \to g_b, d(\epsilon) \to g_d\}$.

The NFA used for the horizontal languages of the HA A_L are:

- $B_{a,q_{a1}} = \langle Q_L, S_{a,q_{a1}} = \{p_b, p_c\}, p_b, \{p_c\}, \Gamma_{a,q_{a1}} = \{(p_b, q_b, p_b), (p_b, q_c, p_c)\}\rangle$,
- $B_{a,q_{a2}} = \langle Q_L, S_{a,q_{a2}} = \{m_b\}, m_b, \{m_b\}, \Gamma_{a,q_{a2}} = \{(m_b, q_b, m_b)\}\rangle$,
- $B_{b,q_b} = \langle Q_L, S_{b,q_b} = \{n\}, n, \{n\}, \Gamma_{b,q_b} = \{\}\rangle$,
- $B_{c,q_c} = \langle Q_L, S_{c,q_c} = \{o\}, o, \{o\}, \Gamma_{c,q_c} = \{\}\rangle$.

$L(A_L) = \{a(bc), a(bbc), \ldots, a(b \ldots bc), \ldots, a, a(b), a(bb), \ldots, a(b \ldots b), \ldots\}$ and $L(A)$ is the set of trees where the root node is labelled with a, where the internal nodes are labelled with b, and where the leaves are labelled with d. We now apply the document adaptation $\upsilon = (REN : b(x) \rightarrow a(x), INS_{first} : c(x) \rightarrow c(g_a x), INS_{before} : c(x) \rightarrow g_a c(x))$, composed of update operations of R/A, and we compute the HA $A' = \langle \Sigma \cup \Sigma_L, P \cup Q_L, Q_L^f, \Delta' \rangle$ such that $L(A') = Post_\upsilon(L)$.

$REN : b(x) \rightarrow a(x)$**;** the NFA $B_{a,q_b} = \langle P \cup Q_L, S_{b,q_b} = \{n\}, n, \{n\}, \Gamma_{b,q_b} = \{\}\rangle$ is defined.

$INS_{first} : c(x) \rightarrow c(g_a x)$**;** the NFA B_{c,q_c} is changed into $\langle P \cup Q_L, \{q_{c,q_c}^{fresh}, o\}, q_{c,q_c}^{fresh}, \{o\}, \{(q_{c,q_c}^{fresh}, g_a, o)\}\rangle$.

$INS_{before} : c(x) \rightarrow g_a c(x)$**;** the NFA $B_{a,q_{a1}}$ is changed into $\langle P \cup Q_L, S_{a,q_{a1}} = \{p_b, p_c, q_{c,q_c}^{fresh}\}, p_b, \{p_c\}, \Gamma_{a,q_{a1}} = \{(p_b, g_a, q_{c,q_c}^{fresh}), (q_{c,q_c}^{fresh}, q_c, p_c), (p_b, q_b, p_b)\}\rangle$.

Figure 2.10 shows an example of application of the document adaptation composed by REN, INS_{first} and INS_{before} rules that transform $t \in L$ into $t' \in L(A')$. The inserted trees of type g_a are $a(b(d))$ and $a(b(d), b(d))$, respectively. Figure 2.11 shows an accepting computation of A' over t'. $\Diamond$

2.6.2 Correctness

Proposition 2.1 discusses the correctness of the proposed algorithm. Given a HA $A = \langle \Sigma, Q, Q_f, \Delta \rangle$, we denote as $t \underset{\Delta}{\rightarrow} q$, $t \underset{\Delta}{\overset{n}{\rightarrow}} q$, $t \underset{\Delta}{\overset{*}{\rightarrow}} q$ the evaluation of a tree $t \in T(\Sigma)$ in state $q \in Q$, in 1, n, an arbitrary but finite number of steps, respectively, using rules in Δ, with $n > 1$.

Proposition 2.1. Let A_L be the HA that denotes the set of documents L, and A' be the HA computed by HASA for the document adaptation υ. Then $L(A') = Post_\upsilon(L)$, where $\upsilon = (r)$ and $r \in \{r\}/A$ (*i.e.*, A is the parameterizing automaton for r). $\square$

Proof. $Post_\upsilon(L) \subseteq L(A')$:
In order to prove that $Post_\upsilon(L) \subseteq L(A')$, we need to prove that for any $t = f(t_1, \ldots, t_k) \in L$, if $t \underset{\Delta_L}{\overset{n+1}{\rightarrow}} q$ and $t \Rightarrow_r t' = f'(t'_1, \ldots, t'_k)$, then $t' \underset{\Delta'}{\overset{m}{\rightarrow}} q$, where $m, n, k \in \mathbb{N}$ and $q \in Q_f$. $t' \in Post_\upsilon(L)$

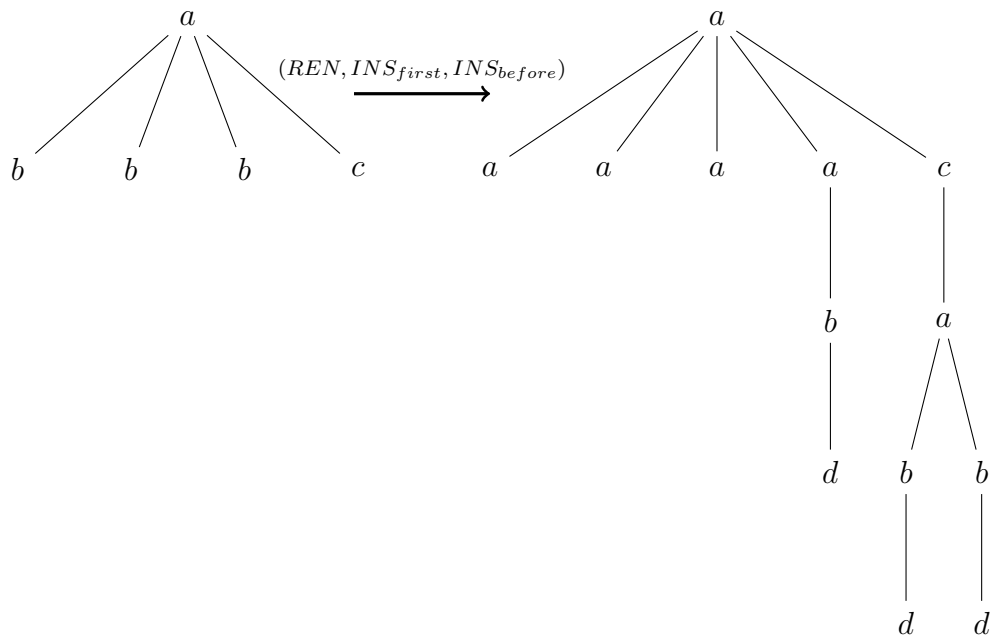


Figure 2.10: The parallel application of REN, INS_{first} and INS_{before} changing $t \in L$ (tree on the left) into $t' \in L(A')$ (tree on the right).

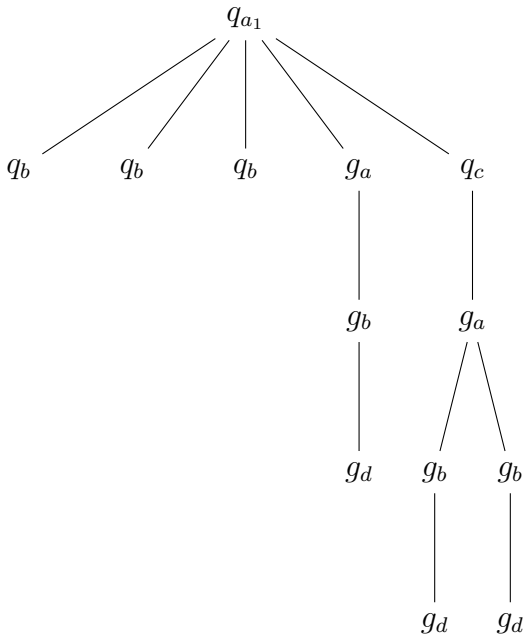


Figure 2.11: The accepting computation of the HA A' over the updated tree t'.

trivially holds, because t' is obtained through parallel rewriting from $t \in L$, using r. At this point we need to prove that $t' \in L(A')$, independently from the type of rule r, by deriving an accepting computation of A' over t' (*i.e.*, it is evaluated into state $q \in Q_f$ by using rules in Δ'), from the corresponding one of A over t (*i.e.*, it is evaluated into state $q \in Q_f$ by using rules in Δ), by arithmetic induction on the length of the latter computation.

- Base case ($n = 0$): $t \underset{\Delta_L}{\longrightarrow} q$, with $t \in \Sigma$. We prove that $t' \in L(A')$ with a case analysis on the type of rule r.

  $r = REN : a(x) \rightarrow b(x)$; then $t = a$ and $t' = b$. By construction of A', any rule $r_L = a(L_a) \rightarrow q_a \in \Delta_L$ has a corresponding rule $r' = b(L_a) \rightarrow q_a \in \Delta'$. Therefore, if $t \underset{r_L}{\longrightarrow} q$, then $t' \underset{r'}{\rightarrow} q$.

  $r = INS_{first} : a(x) \rightarrow a(px)$; then $t = a$ and $t' = a(t'')$. In addition, we have that $t'' \xrightarrow[\Theta]{*} p$ (*i.e.*, $t'' \in L(A)$, by definition), and by construction of A' we have that

  $$\begin{cases} (q^{fresh}_{a,q}, p, f_{a,q}) \in \Gamma'_{a,q} & \text{if } L_{a,q} = \emptyset, \\ \{(q^{fresh}_{a,q}, p, i_{a,q}), (i_{a,q}, y, q_y)\} \subseteq \Gamma'_{a,q} & \text{if } L_{a,q} \neq \emptyset \end{cases}$$

  and $\Theta \subseteq \Delta'$. If $a(L_{a,q}) \rightarrow q \in \Delta_L$ is used as a transition in $t \underset{\Delta_L}{\longrightarrow} q$, then we have that $t' \xrightarrow[\Delta']{*} q$, and the last step is the application of $a(L'_{a,q}) \rightarrow q \in \Delta'$.

  $r = INS_{last} : a(x) \rightarrow a(xp)$; then $t = a$ and $t' = a(t'')$. In addition, we have that $t'' \xrightarrow[\Theta]{*} p$ (*i.e.*, $t'' \in L(A)$, by definition), and by construction of A' we have that

  $$\begin{cases} (i_{a,q}, p, q^{fresh}_{a,q}) \in \Gamma'_{a,q} & \text{if } L_{a,q} = \emptyset, \\ \{(q_y, y, f_{a,q}), (f_{a,q}, p, q^{fresh}_{a,q})\} \subseteq \Gamma'_{a,q} & \text{if } L_{a,q} \neq \emptyset \end{cases}$$

  and $\Theta \subseteq \Delta'$. If $a(L_{a,q}) \rightarrow q \in \Delta_L$ is used as a transition in $t \underset{\Delta_L}{\longrightarrow} q$, then we have $t' \xrightarrow[\Delta']{*} q$ and the last step is the application of $a(L'_{a,q}) \rightarrow q \in \Delta'$.

  $r = INS_{before} : a(x) \rightarrow pa(x)$; then $t = a$ and $t' = t''a$. In addition, we have that $t'' \xrightarrow[\Theta]{*} p$ (*i.e.*, $t'' \in L(A)$, by definition), and by construction of A' and the corresponding assumptions, we have that $\Theta \subseteq \Delta'$ and $\{(s, p, q^{fresh}_{b,q}), (q^{fresh}_{b,q}, q', s')\} \subseteq \Gamma'_{b,q}$, for every rule (s, q', s') in $\Gamma_{b,q}$. So if $b(L_{b,q}) \rightarrow q \in \Delta_L$ is used as a transition in $t \underset{\Delta_L}{\longrightarrow} q$, then we have $t' \xrightarrow[\Delta']{*} q$ and its last step is the application of $b(L'_{b,q}) \rightarrow q \in \Delta'$. Where t appears as a subtree, we have that $\forall i \in [1 \dots n]$. $t'_i \underset{\Delta'}{\longrightarrow} q_i$ and $t'_i(\epsilon) = a$, $q_1 \dots q_{i-1} p q_i \dots q_n \in L'_{b,q}$ iff $q_1 \dots q_{i-1} q_i \dots q_n \in L_{b,q}$.

$r = INS_{after} : a(x) \to a(x)p;$ then $t = a$ and $t' = at''$. In addition, we have that $t'' \xrightarrow[\Theta]{*} p$ (*i.e.*, $t'' \in L(A)$, by definition), and by construction of A' and the corresponding assumptions, we have that $\Theta \subseteq \Delta'$ and $\{(s, q', q^{fresh}_{b,q}), (q^{fresh}_{b,q}, p, s')\} \subseteq \Gamma'_{b,q}$ for every rule (s, q', s') in $\Gamma_{b,q}$. So if $b(L_{b,q}) \to q \in \Delta_L$ is used as a transition in $t \xrightarrow[\Delta_L]{} q$, then we have $t' \xrightarrow[\Delta']{*} q$, and its last step is the application of $b(L'_{b,q}) \to q \in \Delta'$. Where t appears as a subtree, we have that $\forall i \in [1 \dots n]$. $t'_i \xrightarrow[\Delta']{*} q_i$ and $t'_i(\epsilon) = a$, $q_1 \dots q_i p q_{i+1} \dots q_n \in L'_{b,q}$ iff $q_1 \dots q_i q_{i+1} \dots q_n \in L_{b,q}$.

$r = RPL : a(x) \to p;$ then $t = a$ and $t' \xrightarrow[\Theta]{*} p$ (by definition) and, because of $\Theta \subseteq \Delta'$, we have that $t' \xrightarrow[\Delta']{*} p$. Suppose that t appears as a subtree of the root node in a generic tree t_0, where $t_0 \Rightarrow t'_0$. Since by construction of A' we have that $(s, p, s') \in \Gamma'_{b,q}$, for every rule $(s, q, s') \in \Gamma_{b,q}$, we also have that if $b(L_{b,q}) \to q \in \Delta_L$ is the last step of $t_0 \xrightarrow[\Delta_L]{*} q$, in $t'_0 \xrightarrow[\Delta']{*} q$ we will have $b(L'_{b,q}) \to q \in \Delta'$.

$r = DEL : a(x) \to ();$ then $t = a$ and $t' = ()$, but we know that DEL can not be applied to root nodes. Instead, suppose that t appears as a subtree of the root node of a generic tree t_0, where $t_0 \Rightarrow t'_0$. Since by construction of A' we have that $(s, \epsilon, s') \in \Gamma'_{b,q}$, for each rule $(s, q, s') \in \Gamma_{b,q}$, we have that if $b(L_{b,q}) \to q \in \Delta_L$ is the last step of $t_0 \xrightarrow[\Delta_L]{*} q$, in $t'_0 \xrightarrow[\Delta']{*} q$ it will be replaced by $b(L'_{b,q}) \to q \in \Delta'$.

- Inductive case ($n > 0$): Suppose we have that $t_1 \xrightarrow[\Delta_L]{n} t_2 \xrightarrow[\Delta_L]{} t_3$. Then we assume, as inductive hypothesis, that $t'_1 \xrightarrow[\Delta']{*} q_1$, …, $t'_k \xrightarrow[\Delta']{*} q_k$, if $t^1_1 \xrightarrow[\Delta_L]{*} q_1$, …, $t^k_1 \xrightarrow[\Delta_L]{*} q_k$, where $t_1 = f(t^1_1 \dots t^1_k)$. Therefore we have that $t'_1 \xrightarrow[\Delta']{*} t'_2$. At this point, given $t_2 \xrightarrow[\Delta_L]{} t_3$, we obtain $t'_2 \xrightarrow[\Delta']{*} t'_3$ by applying the base case.

$L(A') \subseteq Post_\upsilon(L)$:
By observing that HASA applies in isolation the necessary modifications for each element of the considered document adaptation, we can consider without loss of generality $\upsilon = \{r\}$, that is, a document adaptation composed by a single operation. To prove that $L(A') \subseteq Post_\upsilon(L)$, we start from the observation that the required condition $L(A') \subseteq Post_\upsilon(L)$ is equivalent to $\forall t' \in L(A')$. $\exists t \in L$ such that $t \Rightarrow_r t'$. Let $A'||t'$ to denote an accepting computation of the HA A' over t' (it exists by definition, since $t' \in L(A')$). We now observe that $A'||t'$ is itself a tree, in which nodes are labeled with the accepting states of the corresponding subterms of t'.

Starting from tree $A'||t'$, we have to derive a new computation $A||t$, an accepting computation in A for some t such that $t \Rightarrow_r t'$. Intuitively, the computation (tree) $A||t$ is computed from $A'||t'$

by applying backwards the rules used in the transformation from A to A'. We do not have to exhibit a specific term t, we just need to derive a computation for the set of terms that can be rewritten into t' using υ. More formally, we reason by arithmetic induction on the height of the computation $A'||t'$ of A' over t'.

- Base case ($Height(A'||t') = 1$):

  $r = (REN)$**:** if $A'||t'$ = q_a and two rules $a(L_a) \to q_a \in \Delta_L$, $b(L_b) \to q_b \in \Delta'$ exist such that $\epsilon \in L_a = L_b$, $q_a = q_b$, $a \neq b$, $b(L_b) \to q_b \notin \Delta_L$, and $a(L_a) \to q_a \notin \Delta'$, then the tree $A||t$ corresponds to $A'||t'$, that is, a singleton node with accepting state q_a.

- Base case ($Height(A'||t') = 2$): we first consider an update step applied to a single child of the root node, with a case analysis over the type of the applied update rule r:

  $r = (DEL)$**:** let $A'||t' = q_a(w)$ where k is the maximal natural number such that the following conditions are satisfied

  - $w = q_1, \ldots, q_{i_1}, q_{i_2}, \ldots, q_{i_k}, q_{i_{k+1}}, \ldots, q_n$,
  - $\exists a(L_a) \to q_a \in \Delta_L$, $a(L'_a) \to q_a \in \Delta'$ such that $\exists (s_j, q_{i_j}, s'_j)$, (s'_j, ϵ, s''_j), $(s''_j, q_{i_{j+1}}, s'''_j) \in \Gamma'_{a,q_a}$, $(s'_j, r_j, s''_j) \in \Gamma_{a,q_a}$, for $j \in [1 \ldots k]$.

  Furthermore, assuming that r has form $b(x) \to ()$, then r_j is an accepting state for a subtree having, as root, a b-labeled node, for $j \in [1 \ldots k]$. It is now immediate to check that, from a derivation of the horizontal automaton of L'_a that accepts the string w, we can build an accepting derivation in the horizontal automaton for L_a that accepts the string $w' = q_1, \ldots, q_{i_1}, r_1, q_{i_2}, \ldots, q_{i_k}, r_k, q_{i_k+1}, \ldots, q_n$. Intuitively, we just need to mimic the common steps for rules that are identical in the two automata, and to apply rule (s'_j, q_j, s''_j) in place of (s'_j, ϵ, s''_j), to insert the missing r_j state, for $j \in [1 \ldots k]$. This lemma can be formally proven by arithmetic induction on the length of the derivation in L_a (to take into consideration ϵ transitions). We omit the proof for brevity. From this lemma, we obtain the computation $A||t$, that corresponds to the tree $q_a(w')$. By construction, from any term t accepted by $A||t$ we can derive t' in one step of parallel rewriting. Indeed, any subterm accept by state r_j can be deleted by applying the corresponding DEL rule for $j \in [1 \ldots k]$.

  $r = (RPL)$,$r = (REN)$,$r = (INS_x)$**:** The proof is similar to the first case. The only difference is the pattern searched for replacement in the horizontal languages.

- Inductive case ($Height(A'||t') > 2$): if $A'||t' = q_a(A'||t'_1, \ldots, A'||t'_n)$, by inductive hypothesis we can derive $A||t_1$, ..., $A||t_n$ from $A'||t'_1$, ..., $A'||t'_n$, respectively, and we can also derive $A||t$ by applying the base case ($Height(A'||t') = 2$ on the first two steps of tree $q_a(A||t_1, \ldots, A||t_n)$.

We conclude by observing that, if an accepting computation for a given automaton over a certain tree exists, then that tree necessarily belongs to the language accepted by the automaton itself. Thus, since we have obtained the computation $A||t$ starting from the computation $A'||t'$, and both $t \Rightarrow t'$ and $t \in L(A)$ hold, we have the thesis.

2.6.3 Complexity

In this section we analyze the complexity of the different components of our proposal. First we analyze the impact of the preliminary HA transformation (strong normalization) described in Section 2.6.1. Let $A = \langle \Sigma, Q, Q_f, \Delta \rangle$ be a HA. Given A, let $s_A = |\Sigma|$ be the size of its alphabet, let ht_A be the total number of transitions of the horizontal automata of A, and hs_A the total number of states of the horizontal automata. Suppose that, given a state $q \in Q$, $\forall a \in \Sigma$. $a(L_{a,q}) \rightarrow q$. $\exists b \in \Sigma$. $a \neq b \wedge b(L_{b,q}) \rightarrow q$ holds. Suppose also that q is the only state used in all the horizontal languages and that each symbol of Σ appears as the symbol of one of the operations of the considered document adaptation. This is the worst-case for the HA transformation preliminary to $Post$. In such case the total number of horizontal transitions and the total number of the states of the horizontal automata is increased by $ht_A * s_A$, and by $hs_A + s_A$, respectively.

We now describe the complexity of single or repeated applications of $Post$ computation on a transformed HA A.
Let $o = |\Delta|$ be the number of transitions of A, $p = \arg\max_{a \in \Sigma, s \in Q} |\delta_{a,s}|$ be the maximum number of transitions *per* single horizontal automaton, and $q = \arg\max_{a \in \Sigma} |\{s \mid s \in Q \wedge a(\ldots) \rightarrow s \in \Delta\}|$ be the maximum number of states associated with a symbol. Given the HA A, the time complexity of $Post$, for the different operations, is as follows:

- INS_{first}, INS_{last}: in the worst case we need to scan all the o rules to find the one(s) related to the involved symbol, that are at most p. For each of them we need to create, in constant time, a fresh initial (resp. final) state. The complexity is then in $\mathcal{O}(o + p)$.
- $INS_{after}, INS_{before}, DEL, RPL$: in the worst case we need to scan all the o rules to find the one(s) related to the involved symbol, that are at most q. For each of these q rules we need to traverse the whole set of transitions of the horizontal automaton associated with each rule, in order to delete the old one. The maximum size for the set of horizontal transitions is p. The creation of the new fresh state and the two new transitions is performed in constant time. The complexity is then in $\mathcal{O}(o * p * q)$.
- REN: the disjoint union of each element of the tuple of two horizontal automata can be performed in constant time, while the emptiness test for the horizontal languages is linear in the size of the corresponding NFA.

Therefore, $\mathcal{O}(o * p * q)$ is the higher cost for a single operation. To compute the worst case complexity of n update steps, we first notice that both INS_{before} and INS_{after} can potentially double the current maximum number of (horizontal) transitions, under the following conditions:

- all the transitions of the horizontal automaton with the maximum number of transitions are of the form (s, p, s'),
- the applied rule is of the form $pa(x)$ or $a(x)p$ (that is, an INS_{before} or INS_{after} operation, respectively),
- the considered HA presents a rule of the form $a(z) \to p$, for some sequence of states z.

Let p_i be the maximum number of horizontal transitions for the HA computed at the *(i-1)-th* step, the input for the *i-th* step. Then, the maximum number of horizontal transitions for the HA output of the *i-th* step is

$$p_{i+1} = 2 * p_i = 2 * 2 * p_{i-1} = 2 * 2 * \ldots * 2 * p_0 = \underbrace{2 * 2 * \ldots * 2}_{i+1} * p = 2^{(i+1)} * p.$$

Thus, the *i-th* step operates on $p_i = 2^i * p$ transitions at most. Therefore, a sequence of n steps operates on a number of transitions from p_0 to p_n, where $p_0 = p$. The total complexity is the sum of the cost of the single operations, that is,

$$\sum_{i=0}^{n} o * p_i * q = o * q * \sum_{i=0}^{n} p_i.$$

In turn,

$$\sum_{i=0}^{n} p_i = \sum_{i=0}^{n} (2^i * p) = p * \sum_{i=0}^{n} 2^i = p * (2^{n+1} - 1).$$

So, the total complexity is $o * q * p * (2^{n+1} - 1)$, that is in $\mathcal{O}(p * 2^{n+1} * o * q)$.

However, in realistic updates, it is rare that the whole document adaptation involves the same symbol s (associated with type q_s). In addition, schemas where all the labels accept subtrees of a single type q_s are not common. In realistic schemas, a type usually appears in the definition of multiple elements, but this feature is never employed so massively (*i.e.*, types tend to be locally employed, and are usually not shared by all the elements). Indeed, the distinct rules of realistic document updates, when applied on a realistic schema, affect different horizontal transitions, leading to a limited number of transition duplications. Under this assumption, we provide the complexity for a document adaptation, where a single update adds at most a pair of new transitions. Under this assumptions, the maximum number of horizontal transitions for the HA computed at the *i-th* step, is

$$p_{i+1} = 2 + p_i = 2 + 2 + p_{i-1} = 2 + \ldots + 2 + p_0 = 2 * (i + 1) + p.$$

Thus, the *i-th* step operates on $p_i = 2 * i + p$ transitions at most. The total complexity is equal to

$$\sum_{i=0}^{n} o * p_i * q = o * q * \sum_{i=0}^{n} p_i.$$

In turn,

$$\sum_{i=0}^{n} p_i = p + \sum_{i=1}^{n} p_i = p + \sum_{i=1}^{n} (2*i+p) = p+p+2*\sum_{i=1}^{n} i = 2p+2*\frac{1}{2}(n*(n+1)) = 2p+(n^2+n).$$

Therefore, under this assumptions, the temporal complexity of $Post$, for a given document adaptation υ, is quadratic in the size of υ.[3] Anyway, the temporal complexity of the overall technique dominated by the *Inclusion Test*, that for NFHA is known to be *EXPTIME*-complete [CDG+07, Sei90]. In the next section we experimentally validate our assumptions (*i.e.*, polynomial behavior on real schemas and dominance of the inclusion test), by means of a prototype implementation and examples of schemas taken from real case studies.

2.7 Experimental Evaluation

In this section we present the experimental evaluation of our proof-of-concept implementation. We have developed a Java prototype based on the *LETHAL* library.[4] The $Post$ algorithm implementation works on a representation of the horizontal languages as regular expressions (we adapted our algorithm to deal with it, exploiting the well-known equivalence between regular expressions and NFA) and then computes a new HA. This is due to limitations of the *LETHAL* library, which is not designed for low-level manipulations of automata but only for the application of common HA operations (inclusion, union, intersection, etc.). The tests has been performed on a laptop with an *Intel Core 2 Duo* processor at 2GHz per core (single-threaded implementation), equipped with 4GB of RAM and running *Fedora 17-64bit* as operating system. All the results presented in this section have been averaged over 20 runs.

2.7.1 Impact of Schema Size

In order to confirm the complexity analysis of our symbolic algorithm we performed several performance tests with increasing schema size, for evaluating its computation time. First, we need to clarify the definition of XML schema size. Many metrics have been proposed for defining

[3] For brevity, we did not consider the case in which the parameterizing automaton for the PHRS is different from the automaton describing the document collection. In order to handle this case it is sufficient to consider o as the sum of all the rules of both automata, p and q as the maximum value among the ones of the two automata.

[4] *LETHAL* library is available at `http://lethal.sourceforge.net/`

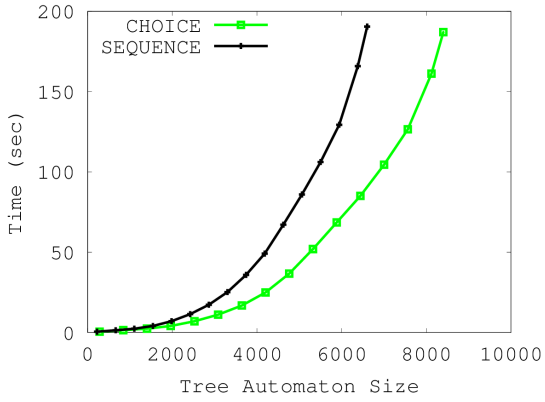


Figure 2.12: Comparison of the computational time of the inclusion test for schemas with increasing automaton size using choice and sequence compositor.

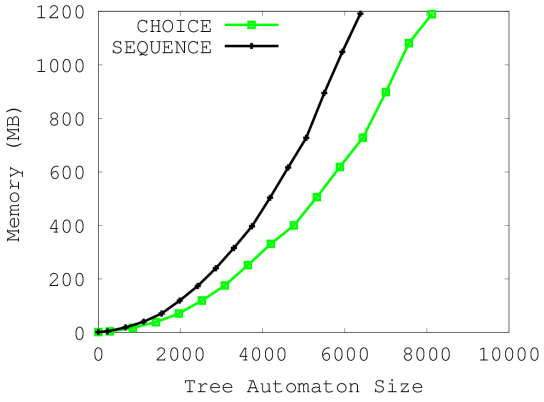


Figure 2.13: Comparison of the memory peak of the inclusion test for schemas with increasing automaton size using choice and sequence compositor.

the complexity or size of an XML schema [MSY04, LKR05, Vis06, BM09]. We decided to employ, as a complexity measure, the size of the TA associated with the HA representing the schema, as done in [CDG$^+$07], Section 1.7. The motivation is that, differently from the measures present in literature, the size of the automaton is strictly related to the complexity of the automata operations employed by our prototype. The size of a TA A is defined as $size(A) = |Q| + \sum_{r \in \Delta}(Arity(symbol(r)) + 2)$.

Given the extreme difficulty in finding the right amount of real schemas with uniformly increasing schema size, and given the quite simple structure of real schema in terms of content model (97% of the content model analyzed in [BNVdB04] are composed of simple expressions only, that is, composed only by simple types), we opted for a generator of synthetic schemas, whose size can be, in this way, easily controlled.

The generator accepts 3 parameters: the number of complex types, the size of the content model and the compositor for the content model, *choice* or *sequence*, that are the XML Schema equivalent to, respectively, *concatenation* and *alternation* operations for regular expressions. In order to ensure a strictly increasing automaton size, in the content model of complex types, we insert elements having a fresh simple type. From a theoretical point of view, two schemas with identical automaton size share the same temporal and spatial complexity, despite the number of types and the compositor used for complex types, the size of the content models, etc. During our experimental evaluation of *LETHAL* library we noticed that the size of the content model (*e.g.*, the number of symbols composing a regular expression representing an horizontal language) has no impact, as well as the number of simple and complex types. That is, schemas with identical automaton size require the same amount of computational time and memory for the inclusion

test. On the other hand, the use of different compositors (*i.e.*, *sequence* and *choice*) changes the computational time and memory peak for the inclusion test, as it is shown in Figure 2.12 and Figure 2.13, respectively.[5] However, this is related to the implementation of the library and that of our schema generator: schemas generated using complex types based on sequence compositor result in "smaller" automata w.r.t. schemas with complex types using choice compositor, using the same content model size and number of types: the last category has an automaton size exceeding the first one by the quantity $(contentSize-1)*|complexTypes|*2+|complexTypes|*|simpleTypes|$. With the sequence compositor, indeed, only the last element has a transition to the final state, while with choice compositor all the others $(contentSize-1)$ have a similar transition, and this situation holds for each complex type present in the schema (their number is identified by the term $|complexTypes|$ in the previous formula). Moreover, each of these transitions is weighted 2 in the formula computing the TA size. In addition, due to the determinization phase of the horizontal automata, choice compositor needs additional intermediate states that can be quantified as $|complexTypes|*|simpleTypes|$. For this reason, in order to "achieve" the same automaton size, the schemas based on sequence compositor need a higher number of HA rules, thus imposing a computational overhead on the creation and manipulation of the data structures describing the automata, as well as increasing their memory occupation. Figure 2.12 and Figure 2.13 also show that the inclusion test requires exponential time and polynomial memory in the automaton size.

2.7.2 Impact of Update Sequence

The effect of the length of the document adaptation and its composition (in terms of involved *XQUF* primitives) has been empirically determined with another test suite. In literature there are neither benchmarks for updates nor for schema update sequences, and it is difficult to define the shape of realistic update sequence. For this reason, as in related approaches aiming at evaluating schema updates [GMS07, NKMM12], we rely on synthetic update sequences. Specifically, we employ a random sequence generator, generating random sequences composed by insert operations only, since the worst case time complexity for the HASA module involves insertions. In addition, insertions always increase schema size and ensure a non-decreasing complexity, even for long sequences (*e.g.*, a deletion of complex type directly used in the content model of the starting symbol of the tree grammar, or its replacement with a simple type, may significantly reduce the corresponding automaton size). In Figure 2.14 and Figure 2.15, the impact of increasing sequence length on the computational time and memory occupation are shown, respectively.[6] Figure 2.14 confirms the theoretical evaluation of the time complexity of the HASA algorithm presented in Section 2.6: its complexity is dominated by the exponential behavior of the inclu-

[5]The tests presented in Figure 2.12 and Figure 2.13 employ a number of complex types ranging from 1 to 150, with a content model size of 5 elements of distinct simple types (from a minimum of 5 up to a maximum of 750).

[6]The tests presented in Figure 2.14 and Figure 2.15 employ an update sequence length ranging from 4 to 2400, equally composed by the four insertion operations, each with a randomly chosen target.

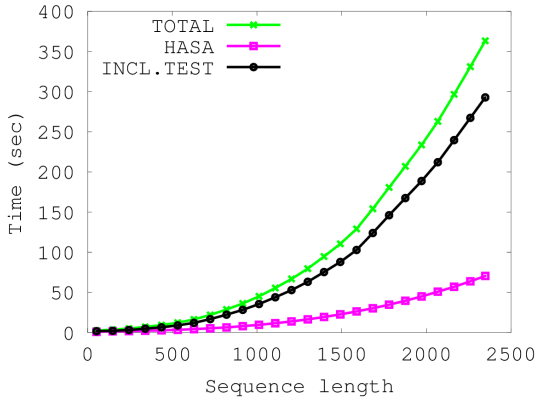


Figure 2.14: Computational time of the algorithm with an increasing update sequence length composed of insertion operations only.

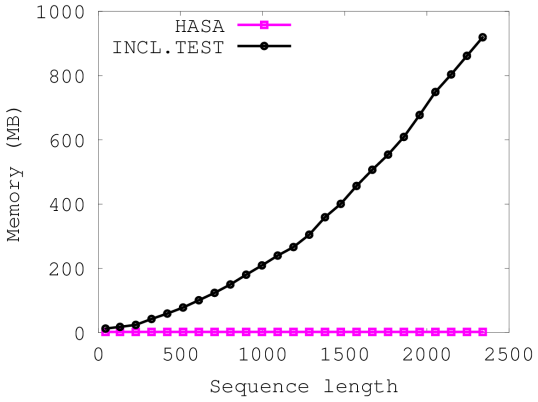


Figure 2.15: Memory occupation peak of the algorithm with an increasing update sequence length composed of insertion operations only.

sion test. The size of the automata, however, does not grow exponentially as in the worst-case analysis, but it presents a polynomial behavior. This is related to the fact that real schemas, like the $XMarks$ one employed in this test[7] [S+01], have an associated HA that does not present horizontal languages with a massive repetition of the same transition label. In synthesis, the worst-case exponential behavior of $Post$ computation does not seem to have an impact. In practice, $Post$ has instead a quadratic behavior. In Figure 2.15, the memory occupation peaks of the HASA module and the inclusion test are shown: the HASA module presents a linear space occupation (constantly around 2MB) and it is dominated by the polynomial space required to perform the inclusion test: so the minimum memory required for the algorithm execution is equal to the memory peak reached during the inclusion test.

2.7.3 Practical Usage Evaluation

Inclusion test complexity for NFHA is *EXPTIME*-complete [Sei90, CDG+07]. This operation dominates the complexity of our module and poses some limitations to its the practical usage. The automaton size depends on the corresponding schema size, that is usually limited (in terms of labels and productions) in real-world schemas. The authors of [MTP06], for example, report that most of the documents analyzed in their study (around 16534 schemas) show a quite simple structure, using only a very limited number of distinct element and attribute names (usually less than 150). In addition, schema size in not comparable with the one of the associated document

[7]The benchmark and related schema are available at `http://www.xml-benchmark.org/`

collection (in terms of both document number and size). The tests show the potential of our proposal for a practical usage as a support for static analysis of XML document adaptations.

Chapter 3

Synthetising Changes in XML Document as Pending Update Lists

3.1 Introduction

The ability of detecting changes in documents, through the so called *diff* algorithms, is crucial in many data management contexts. Such algorithms take as input two pieces of data and detect the differences between them, so that such differences can be analyzed or employed to tranform data. Various representations can be adopted for differences, usually referred to as *edit-scripts* or *deltas*.

The problem has been widely investigated for flat text files as well as for hierarchically-structured data and XML documents. For flat text files, the popular GNU *diff* utility, based on the *LCS* algorithm for finding the longest common subsequences in strings [MM85, Mye86], works well. More and more often, however, information is hierarchically structured and represented in XML. As a consequence, there has been a relevant number of studies devoted to the detection of changes in hierarchically-structured documents in general [Cha99, CRGMW96] and in XML documents [CAM02, Fon01, WDC03, XWW+02] specifically. Such approaches take the structural properties imposed by the tag nesting into account. They differ in the assumptions on the tree model (*e.g.*, ordered or unordered), in the format of the edit-scripts, in the cost models, in the assumptions on the document contents (*e.g.*, duplicate subtrees are uncommon and can be used to identify reliable matches). Another significant difference in the proposed algorithms is the chosen trade-off between result optimality and algorithm complexity.

The standard language for updating XML documents is the W3C recommendation XQuery Update Facility (XQUF) [W3C11]. The evaluation of an XQUF expression on a document produces an unordered list of atomic update requests, represented as a *Pending Update List* (PUL), that

is then applied on the document. A decoupled PUL execution model, by which PULs can be exchanged across machines and applied on a machine different from the one where they have been produced, has been proposed [CGM11a].

An XML differencing algorithm is only one of the tools in an XML engineer toolbox. In most contexts where change detection is useful, such as collaborative editing or versioning, other change manipulation operations are needed. Notable examples are change reverting, merging, reconciliation, and composition of a sequence of consecutive changes. Operators for these manipulations on PULs have been proposed in [CGM11a] and implemented in the Zorba[1] XQUF engine.

PULs are thus an excellent candidate as edit-script language and we believe that a PUL-based, fast, and efficient algorithm for differencing XML documents is needed. Unfortunately, the XML diff algorithms proposed in the literature cannot be easily adapted due to the supported set of edit primitives (*e.g.*, there is no move operator in PULs, while relabelings and changes at leaves can be detected) or the assumptions they rely on. Another limitation of these sets of edit operations w.r.t. PULs is that the order of update operations of the same kind is not prescribed, and all update operations refer to the original document.

Outline. The chapter is organized as follows. Section 3.2 discusses relevant related work, whereas Section 3.3 introduces some preliminary notions. Section 3.4 presents the PUL-Diff algorithm. Section 3.5 and Section 3.6 detail its relevant components, namely, the identification of the heaviest increasing point subset and the PUL tree edit-distance, respectively. Section 3.7 provides an evaluation of PUL-Diff (*i.e.*, complexity analysis, experiments, and comparative analysis with state of the art XML differencing algorithms).

3.2 Related Work

Given two trees and a set of operations with an associated cost model, the tree edit-distance problem is to compute the minimum-cost sequence of operations that transforms the first tree in the second one. The best known algorithms for the tree edit-distance problem decompose the trees into smaller subtrees and subforests by removing one node after the other. The classical algorithm by Zhang and Shasha [ZS89] runs in $\mathcal{O}(n^4)$ time, whereas the proposal of Demaine *et al.* [DMRW09] is worst-case optimal and runs in $\mathcal{O}(n^3)$. The runtime behavior of these algorithms heavily depends on the tree shapes and on the employed decomposition strategy. At each decomposition step, they rely on a fixed strategy to choose which node to remove. Pawlik and Augsten [PA11] dynamically and optimally choose a different strategy. The proposed algorithm runtime complexity is in $\mathcal{O}(n^3)$ and outperforms the previous proposals. Given the high computational cost of the exact approaches, many approximated algorithms have been proposed.

[1] `http://www.zorba-xquery.com`

The MH-Diff algorithm [CGM97], targeting unordered trees, is a heuristic-based approach, where the edit-distance problem is reduced to a minimum cost edge cover in a bipartite graph. Each operation is given a user-defined fixed cost, except for the relabeling operation that employs a user-provided function that compares the values of two nodes. XyDiff [CAM02] computes hash values for all the subtrees of the analyzed trees using DOMHASH,[2] an efficient hashing function specifically tailored for XML subtrees. XyDiff then searches for exact matches in a bottom-up traversal and eagerly tries to expand them looking for common ancestors for the two trees, relying on the node names. This behavior helps detecting renaming, but using hash values and bottom-up traversal prevents the detection of changes in the leaves, a very frequent kind of update. Matches are then improved through a top-down visit of the two trees, trying to propagate existing matches. X-Diff addresses change detection for XML documents modeled as unordered trees [WDC03]. It integrates key XML structure characteristics with a standard edit-distance technique, thus resulting in an efficient algorithm. The algorithm computes hashes similarly to XyDiff but with XHash, that is suited to commutative models. This algorithm considers our set of primitive operations. Kf-Diff [XWW+02] achieves linear time-complexity transforming tree-to-tree correction problem into tree comparison without duplicated paths. It supports both ordered and unordered trees. The algorithm is label-based and thus not robust against renaming. To the best of our knowledge there are no proposals tailored to W3C's PULs.

Augsten *et al.* [ABG10] propose an estimation of the tree edit-distance for ordered trees based on *pq*-grams, subtrees of a fixed shape corresponding to the substrings (called *q*-grams) used for string similarity evaluation. *pq*-grams are composed by a stem made of p elements (bound by the parent-child relation) and a base of q consecutive siblings: the last element of the stem is named anchor node. A tree is therefore represented by its set of *pq*-grams like a string is represented by its *q*-grams. *pq*-grams have been successfully used in different application contexts. The authors of [LWH+14] employed *pq*-grams for estimating XML fragments similarity (considering both tree structure and label similarity) in the context of an approximate join algorithm for XML. Scalability is achieved by using a Min-hash-based method for efficient estimation of the distance between trees and a sort-merge hash similarity join algorithm.

pq-grams and their variants have also been used in [CO14], in the context of subtree similarity-search problem (also known as approximate subtree matching), which is the problem that aims at finding the most similar candidates for a given tree, among the subtrees of forest. The proposed technique, that exploits dynamic programming for the exploration of the solution space, is general and can be applied to different tree-distance functions, for both ordered and unordered trees.

The use of PULs as an edit-script language requires a special care in the use of *pq*-grams. We therefore propose an adaptation of *pq*-grams where in each *pq*-gram the nesting depth of its anchor node is included (in order to reflect the absence of a move operation in PULs), and where different *pq*-grams for node names and values are used (not to overestimate PUL tree edit-

[2] `http://www.ietf.org/rfc/rfc2803.txt`

distance when nodes have different names or values). The absence of a move primitive among PULs operations also requires a method for removing matches that would be expressed through this operation. This problem can be reduced to a generalization of the *Heaviest Increasing Subsequence* (HIS) problem [JV92]. Such generalized *Heaviest Increasing Point Subset* (HIPS) problem aims at finding, given a set of two-dimensional weighted points, one of the maximal-weight subsets defining a strictly increasing function.

3.3 Preliminaries

In this section, we introduce the definitions and concepts used in the remainder of the chapter. Specifically, the tree representation of XML documents is introduced in Section 3.3.1, the XQUF language in Section 3.3.2, while its dynamic representation of updates (*i.e.*, PULs) is introduced in Section 3.3.3. As a general remark we stress that, despite the partial overlapping with the data structures (XML documents and their tree representation) and associated operations (XQUF and PULs) used in Chapter 2, here the abstraction level is lower and more system-oriented than in the previous chapter. For instance, while in Chapter 2 an XML document is conceptualized as an algebraic structure, here we consider it as a concrete file. Again, where in Chapter 2 we model PUL primitives as hedge rewriting system rules, here their operational semantics is given in terms of side effects over the concrete XML file representation. For this reason, in order to avoid possible confusions, we briefly re-introduce the necessary notions, and we also employ slightly different notations to differentiate corresponding, but not identical, concepts.

3.3.1 Tree Representation of XML Documents

The data model of XQuery is XDM,[3] which describes all permissible values of expressions in the language. Every instance of the data model is a sequence, that is, an ordered collection of zero or more items. An item is either a node or an atomic value. Nodes can be nested to form a tree and are of different kinds. For brevity we consider only document, element, and text nodes. Document nodes do not have a parent, while only element and document nodes may have children. The textual contents of elements are modeled by separate text nodes. A *document* is a tree whose root node is a document node. Given a tree T, we denote its root as $\mathcal{R}(T)$, its nodes as $\mathcal{V}(T)$, and its weight (that is, the number of its nodes) as $\Omega(T)$. The weight of a node v, denoted as $\Omega(v)$, is equal to the weight of the subtree rooted at that node, denoted as $\mathcal{T}(v)$. The weight of a list of trees L, denoted as $\Omega(L)$, is equal to the sum of the tree weights. The empty list is denoted as Λ. Given a node $v \in \mathcal{V}(T)$, we denote its children as $\gamma_E(v)$, its ancestors as $\mathcal{A}(v)$, and its parent as $\mathcal{P}(v)$. The $<_p$ relation corresponds to the document order, that is, the ordering of nodes in the XML serialization of a document.

[3] `http://www.w3.org/TR/xpath-datamodel`

3.3.2 XQUF and PULs

An XQuery expression is evaluated on one or more XDM instances and returns an XDM instance. Its evaluation does not change the state (that is, the parent, children, name, or value) of any node. XQUF extends the language by introducing a new kind of expressions, named *updating expressions*, that can alter the state of nodes, and a set of low level *update operations* representing a node state change. The evaluation of an updating expression results in a single *pending update list* (PUL), that is, an unordered list of update operations. Two phases can thus be identified in the evaluation of an XQUF expression: (i) *production*, in which the expression is evaluated producing a PUL; (ii) *application*, in which the update operations in the produced PUL are applied. The application order is dictated by the operation kind and is specified in [W3C11]. Let v and v' be nodes, and let $L = [T_1, \ldots, T_k]$ be a list (which can be empty, for the `repN` operation) of trees. The PUL node update operations are the following:

- (i) $\texttt{ins}^{\leftarrow}(v, L)$, $\texttt{ins}^{\rightarrow}(v, L)$, $\texttt{ins}^{\swarrow}(v, L)$, $\texttt{ins}^{\searrow}(v, L)$ insert the tree sequence L before/after or as first/last child of v, respectively;
- (ii) $\texttt{ins}^{\downarrow}(v, L)$ inserts the tree sequence L as child of node v, in an implementation dependent position;
- (iii) $\texttt{del}(v)$ detaches node v from its parent (if any), and the subtree rooted at v becomes a new tree;
- (iv) $\texttt{repN}(v, L)$ replaces node v with the tree sequence L, and the subtree rooted at v becomes a new tree;
- (v) $\texttt{repV}(v, s)$ replaces the value of node v with the value s;
- (vi) $\texttt{repC}(v, v')$ replaces the children of node v with the optional text node v';
- (vii) $\texttt{ren}(v, n)$ renames node v as n.

We do not consider `put` operation, as it is not a node update operation, and `insertAttributes` operation, as we do not consider node attributes.

We assign to each operation a fixed unitary cost, to which we sum the weight of the removed/inserted nodes, as formally defined in Definition 3.1.

Definition 3.1. Let op be an *operation*. Its *cost*, denoted as $\xi(op)$, is:

$$\xi(op) = \begin{cases} 1 + \Omega(L) & \text{if } op = ins^d(v, L), d \in \{\leftarrow, \rightarrow, \swarrow, \searrow, \downarrow\} \\ 1 + \Omega(v) & \text{if } op = \texttt{del}(v) \\ 1 & \text{if } op \in \{\texttt{repV}(v, s), \texttt{ren}(v, n)\} \\ 1 + \Omega(v) + \Omega(L) & \text{if } op = \texttt{repN}(v, L) \\ \Omega(v) + \Omega(v') & \text{if } op = \texttt{repC}(v, v'), \\ & \text{where } \Omega(\Lambda) = 0 \end{cases}$$

△

Finally, the cost of a PUL, introduced in Definition 3.2, is defined as the sum of the cost of each operation composing the PUL itself.

Definition 3.2. Let Δ be a *PUL*. The *cost* of Δ, denoted as $\xi(\Delta)$, is equal to $\sum_{op \in \Delta} \xi(op)$. △

3.3.3 Matches and Weights

The PUL-Diff algorithm (presented in Algorithm 1) compares two XML documents, named source (resp. target), and denoted as S (resp. T). Subscripts s and t denote nodes in the source and target trees, respectively, and are omitted when no confusion arises. This comparison has one main goal: decide which subtrees of the source document should be considered deleted (*i.e.*, present in the source document but not in the target one), which should be considered inserted (*i.e.*, present in the target document but not in the source one), and which should be considered *matching*, that is, present in both documents with marginal or no changes. In the former case the edit-script expresses the transformation of these matching trees. The concept of (complete) node match is expressed in Definition 3.3.

Definition 3.3. A *match* m is a pair $\langle s, t \rangle$, where $s \in \mathcal{V}(S)$ and $t \in \mathcal{V}(T)$. A match m is complete if $\mathcal{T}(s)$ is identical to $\mathcal{T}(t)$, partial otherwise. △

Intuitively, the weight of a match $\langle s, t \rangle$, introduced in Definition 3.4, measures the advantage, in terms of edit-script cost, of considering the two nodes to match over considering them not to match. The match weight thus ranges from $1 + \Omega(s) + \Omega(t)$ when the two subtrees are identical (no operation required if we consider them to match, $\texttt{repN}(s, t)$ if we consider them not to match) to 0 when s and t are so different that the most cost-efficient transformation is $\texttt{repN}(s, t)$, that replaces s with t.

Definition 3.4. The *weight of a match* $\langle s, t \rangle$, denoted as $\psi(\langle s, t \rangle)$, is defined as $1 + \Omega(s) + \Omega(t) - \xi(\Delta)$, where Δ is a minimum cost edit-script transforming $\mathcal{T}(s)$ in $\mathcal{T}(t)$. △

Given a set of matches M, function ψ returns the sum of the weights of all the matches in M. Let $\langle s, t \rangle$ be a match. Since no move operation is allowed in a PUL, it may not be possible to transform $\mathcal{T}(s)$ into $\mathcal{T}(t)$, without removing $\mathcal{T}(s)$ and inserting $\mathcal{T}(t)$. If we consider each

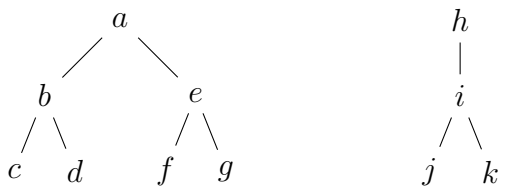


Figure 3.1: Trees A (left) and B (right) used in Example 3.1.

match in isolation this situation happens whenever s and t are at different nesting depths in the respective trees. This notion of valid match is given in Definition 3.5.

Definition 3.5. A *match* $\langle s, t\rangle$ *is valid* iff s and t are at the same nesting depth. $\triangle$

If we consider a set of matches, we need to ensure that, for each pair of matches, the two source and target nodes are in the same structural relationship and that the predecessor relationship is preserved, as formally expressed in Definition 3.6, where compatible matches are defined.

Definition 3.6. Let $\langle q, r\rangle$ and $\langle s, t\rangle$ be two valid matches. $\langle q, r\rangle$ and $\langle s, t\rangle$ are *compatible* iff all of the following conditions hold:

(i) $(q <_p s \iff r <_p t)$,

(i) $(q \in \mathcal{A}(s) \iff r \in \mathcal{A}(t))$,

(i) $(\mathcal{P}(q) = \mathcal{P}(s) \iff \mathcal{P}(r) = \mathcal{P}(t))$.

$\triangle$

Different instances of compatible and not compatible matches are given in Example 3.1.

Example 3.1. Let $A = a(b(c, d), e(f, g))$ and $B = h(i(j, k))$ be two trees represented as terms (*i.e.*, A has an a labeled root node, with b and e labeled children; node b has in turn c and d labeled children). According to Definition 3.6, the following pairs of matches between nodes of A and B are not compatible:

- $\langle c, k\rangle$ and $\langle d, j\rangle$ (violation to condition (i)),
- $\langle c, j\rangle$ and $\langle e, i\rangle$ (violation to condition (ii)),
- $\langle c, j\rangle$ and $\langle g, k\rangle$ (violation to condition (iii)).

Finally, $\langle c, j\rangle$ and $\langle d, k\rangle$ are compatible matches. $\Diamond$

Algorithm 1 PUL-Diff Algorithm

```
1: function PUL-DIFF(S,T)
2:     matches ← ∅
3:     identicalSubtreeMatching(R(T), ref matches)
4:     if matches[R(T)] ≠ Λ then
5:         return ∅
6:     end if
7:     if matches = ∅ then
8:         matches[R(T)] ← {⟨R(S), R(T), 0, False⟩}
9:     else
10:        refineBottomUp(R(T), ref matches)
11:    end if
12:    refineTopDown(R(S),R(T), matches, ref matches)
13:    if matches = ∅ then
14:        return {repN(R(S), R(T))}
15:    end if
16:    return generateEditScript(R(S), R(T), matches)
17: end function
```

Starting from the definition of compatible matches, in Definition 3.7 the notion is extended to set of matches.

Definition 3.7. A *set M of matches is consistent* iff all its elements are pair-wise compatible. $\triangle$

A positive and negative instance of consistency for a set of matches is given in Example 3.2.

Example 3.2. Let $S = \{\langle 1,1,1\rangle, \langle 2,3,1\rangle, \langle 3,3,1\rangle, \langle 3,5,1\rangle, \langle 6,6,1\rangle\}$ be a set of matches. S is not consistent because $\langle 2,3,1\rangle$ and $\langle 3,3,1\rangle$ are not compatible. $\{\langle 1,1,1\rangle, \langle 2,3,1\rangle, \langle 3,5,1\rangle, \langle 6,6,1\rangle\}$, instead, is a consistent subset of S. $\Diamond$

3.4 Algorithm

PUL-Diff compares a source and a target document and produces an edit-script that reflects the difference between them. The edit-script can then be used to transform the source document into the target one. In the edit-script, the subtrees which are present in only one of the two documents will be either inserted or deleted, while similar matching subtrees will be transformed one into the other (*e.g.*, by means of renaming operations). Identical matched subtrees, instead, will not require any modification. Since PULs do not allow us to "move" subtrees without deleting and re-inserting them, not all the identical or similar subtrees of the two documents should be matched. PUL-Diff first detects and matches the similar or identical subtrees of the two documents and then decide which matches should be kept and which should be discarded to obtain a consistent match set. To guide this decision, with the objective of reducing the edit-script cost, we rely on the match weight.

PUL-Diff is structured in four subsequent stages: (i) identical subtree matching, (ii) bottom-up refinement, (iii) top-down refinement, and (iv) edit-script generation. Ideally, the goal of the

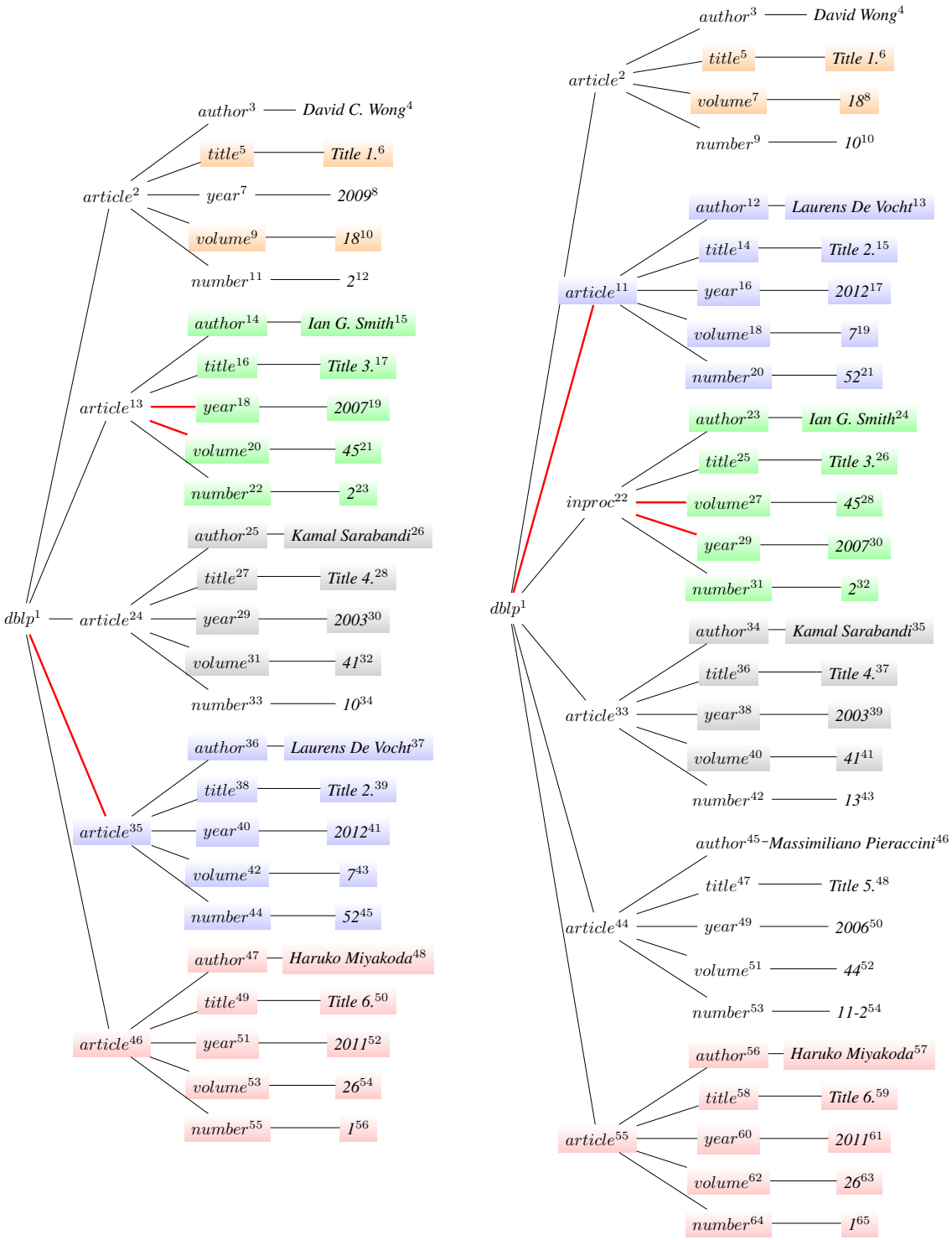


Figure 3.2: An example of source (left) and target (right) trees. We highlight with the same color corresponding parts of the two documents.

Algorithm 2 Identical Subtree Matching

```
1: function identicalSubtreeMatching(t, ref matches)
2:     candidates ← identicalST(t)
3:     if |candidates| = 1 then
4:         matches[t] ← ⟨candidates[0], t, 1 + Ω(s) + Ω(t), True⟩
5:     else if |candidates| = 0 then
6:         for each d in γ_E(t) do
7:             identicalSubtreeMatching(d, ref matches)
8:         end for
9:     end if
10: end function
```

first three stages is to detect the heaviest consistent subset of the set of all possible matches $M = \{\langle s, t\rangle \mid s \in \mathcal{V}(S), t \in \mathcal{V}(T)\}$, that is, detect the best possible matches between the source and target trees. Since this computation would be extremely expensive we employ the following heuristics.

In the identical subtree matching stage (Section 3.4.1), the algorithm looks for complete valid matches. In this stage, we only consider subtrees appearing once in the source document. The purpose of the bottom-up refinement stage (Section 3.4.2) is to detect the best suitable match for any unmatched target node having at least one matched descendant, while ensuring that all detected matches are compatible. In the top-down refinement stage (Section 3.4.3) the algorithm tries to match unmatched descendants of matched target nodes. In the edit-script generation stage (Section 3.4.4) the two trees are visited in a bottom-up fashion for generating an edit-script, according to the previously detected matches.

The detected matches are stored in a map named $matches$, which associates nodes of the target tree with their matches. Since match weights will often be estimated, the representation of a match $\langle s, t\rangle$ (Definition 3.3) is extended to a quadruple $\langle s, t, w, c\rangle$, where w is an estimation of $\psi(\langle s, t\rangle)$ and $c \in \{True, False\}$ specifies whether the match is complete or not.

In the presented algorithms the `ref` keyword denotes function arguments passed by reference, both in function calls and within function signatures. In what follows we detail the four stages composing the PUL-Diff algorithm.

3.4.1 Identical Subtree Matching

The identical subtree matching stage, described in Algorithm 2, matches identical subtrees of the two documents. These matches will be used to limit the matches search effort in the following stages. For this reason we do not consider significant (*i.e.*, we do not add them to the matches map) those matches involving a source subtree that is repeated in the source document. Tree comparison is hash-based. Before the first invocation of Algorithm 2, we compute, for each node n in the source or target tree, a DOMHASH-like signature that identifies the subtree rooted at n and the nesting depth of n. Given a node t, the function identicalST(t) returns each node s

Target Node	Depth	Stage 1 Matches	Stage 2 Matches	Stage 3 Matches
$dblp^{1}$	0		$dblp^{1}$	$dblp^{1}$
$article^{2}$	1		$article^{2}$	$article^{2}$
$author^{3}$	2			$author^{3}$
$title^{5}$	2	$title^{5}$	$title^{5}$	$title^{5}$
$volume^{7}$	2	$volume^{9}$	$volume^{9}$	$volume^{9}$
$number^{9}$	2	$number^{33}$		$number^{11}$
$article^{11}$	1	$article^{35}$		
$author^{12}$	2	↑		
$title^{14}$	2	↑		
$year^{16}$	2	↑		
$volume^{18}$	2	↑		
$number^{20}$	2	↑		
$inproc^{22}$	1		$article^{13}$	$article^{13}$
$author^{23}$	2	$author^{14}$	$author^{14}$	$author^{14}$
$title^{25}$	2	$title^{16}$	$title^{16}$	$title^{16}$
$volume^{27}$	2	$volume^{20}$	$volume^{20}$	$volume^{20}$
$year^{29}$	2	$year^{18}$		
$number^{31}$	2			$number^{22}$
$article^{33}$	1		$article^{24}$	$article^{24}$
$author^{34}$	2	$author^{25}$	$author^{25}$	$author^{25}$
$title^{36}$	2	$title^{27}$	$title^{27}$	$title^{27}$
$year^{38}$	2	$year^{29}$	$year^{29}$	$year^{29}$
$volume^{40}$	2	$volume^{31}$	$volume^{31}$	$volume^{31}$
$number^{42}$	2			$number^{33}$
$article^{44}$	1			
$author^{45}$	2			
$title^{47}$	2			
$year^{49}$	2			
$volume^{51}$	2			
$number^{53}$	2			
$article^{55}$	1	$article^{46}$	$article^{46}$	$article^{46}$
$author^{56}$	2	↑	↑	↑
$title^{58}$	2	↑	↑	↑
$year^{60}$	2	↑	↑	↑
$volume^{62}$	2	↑	↑	↑
$number^{64}$	2	↑	↑	↑

Table 3.1: Summary of the matches detected during the three stages of PUL-Diff algorithm, when applied to source (left) and target (right) trees of Figure 3.2. The first column lists all target nodes, whereas the second column reports their nesting depth. The remaining columns contain the matched nodes, as detected by the PUL-Diff at the end of each of the first three stages. The ↑ symbol denotes that an ancestor of the node has a complete (*i.e.*, identical) match.

of the source tree which is at the same nesting depth of t, and whose subtree is identical to the subtree of t.

An example of identical subtree matching stage is given in Example 3.3.

Example 3.3. Consider the source and target trees in Figure 3.2. The matches detected during the first stage of the algorithm are reported in Table 3.1. Note that no matches have been detected for node $number_t^{31}$, since its subtree is repeated in the source tree. $\Diamond$

3.4.2 Bottom-up Refinement

The bottom-up refinement stage, detailed in Algorithm 3, solves all incompatibilities among the matches detected in the previous stage and tries to match all unmatched target nodes with at least one matched descendant.

Specifically, we perform a bottom-up visit of the target tree and for every visited unmatched node t we check whether any of its children has been matched. In case none has been matched, we do not have enough information to propose a match for t at this stage. Otherwise, we consider the parent of each node matching one of the children of t to be a candidate for matching t. The algorithm then chooses one of the candidates, say s, and produces a match between s and t, discarding all matches among children of other candidates and children of t. Moreover, the algorithm may also discard some of the matches among children of s and children of t, if some of them are incompatible.

Choosing the best candidate for matching t is crucial for the quality of the generated edit-script. Ideally, we would like to compare all matches among t and each candidate and choose the match with the highest weight. However, since determining match weights requires to solve the exact tree edit-distance problem, we need to approximate its computation. A first approximation is to determine, for each candidate node $cand$, the set of matches $C_{cand} = \{\langle c, d, w, r\rangle \mid \langle c, d, w, r\rangle \in matches \wedge c \in \gamma_E(cand) \wedge d \in \gamma_E(t)\}$ among its children and the children of t, and to consider the sum $W_{cand} = \sum_{\langle c,d,w,r\rangle \in C_{cand}} w$ of all the weights of the matches in C_{cand} as an estimation of the match weight between $cand$ and t. This approximation follows the observation that, if we consider $cand$ and t to be matching, all matches of the children of t that are not in C_{cand} would be discarded, while at least one of the matches among the children of $cand$ and children of t would be kept. The main limitation of this estimation is that incompatible matches may be present in C_{cand} and, if this happens, the real match weight of $cand$ and t could be lower than estimated, increasing the chance of choosing a suboptimal candidate. We refine the approximation considering the weight of one of the heaviest consistent subset HC_{cand} of C_{cand}. This problem can be solved using the HIPS algorithm, as we detail in Section 3.5. Then we select the candidate s whose heaviest consistent subset has the highest weight (function getBest at line 12, Algorithm 3) and we add the match of s with t. Finally, all matches in $C_{cand} \setminus HC_{cand}$ and all matches among the children of other candidates and children of t are removed, since they are

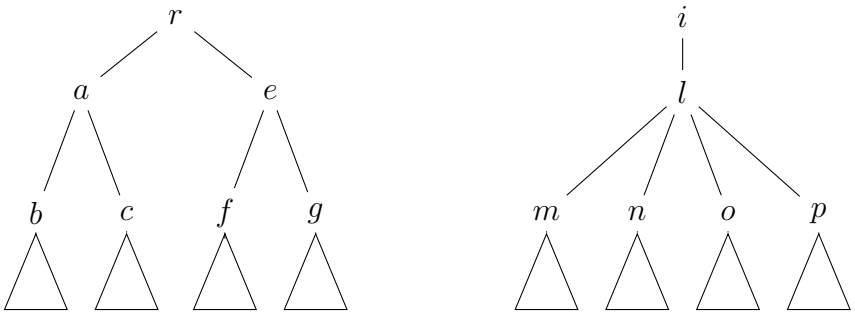


Figure 3.3: Example 3.5 trees A (left) and B (right).

incompatible with the match of s and t.

Example 3.4 shows the bottom-up refinement stage in action.

Example 3.4. Consider the source and target trees in Figure 3.2, and the detected matches (identified at stage 1) reported in Table 3.1. In the bottom-up refinement stage, new matches are detected for the target nodes $dblp_t^1$, $article_t^2$, $inproc_t^{22}$ and $article_t^{33}$, whereas the match of $number_t^9$ and $year_t^{29}$ are discarded. Consider node $article_t^2$. Two candidate nodes are identified: $article_s^{24}$ (due to the match $\langle number_s^{33}, number_t^9, 2, True\rangle$) and $article_s^2$ (due to the matches $\langle title_s^5, title_t^5, 2, True\rangle$ and $\langle volume_s^9, volume_t^7, 2, True\rangle$). The algorithm matches $article_s^2$ and $article_t^2$ and removes the match $\langle number_s^{33}, number_t^9, 2, True\rangle$). Consider now $inproc_t^{22}$. A single candidate is identified (that is, $article_s^{13}$). The set of matches among the children of node $inproc_t^{22}$ and the children of node $article_s^{13}$ however is not consistent as $\langle year_s^{18}, year_t^{29}, 2, True\rangle$ and $\langle volume_s^{20}, volume_t^{27}, 2, True\rangle$ are not compatible. The algorithm matches $article_s^{13}$ and $inproc_t^{22}$, discarding the match $\langle year_s^{18}, year_t^{29}, 2, True\rangle$. $\Diamond$

In Example 3.5 we show how the identification of a consistent set of matches for the children of the candidate nodes can (positively) influence the choice of the new match.

Example 3.5. Consider the trees A and B reported in Figure 3.3 and assume that the following matches have been detected by the identical subtree matching stage: $\langle c, m, 5, True\rangle$, $\langle b, n, 7, True\rangle$, $\langle f, o, 8, True\rangle$. We identify two candidate nodes for matching l: a and e. The estimated weight of the match among e and l is 8. Since $\langle c, m, 5, True\rangle$ and $\langle b, n, 7, True\rangle$ are not compatible, the weight of the match among a and l is estimated as 7. The algorithm thus produces a new match $\langle a, l, 8, True\rangle$ and discards the two matches $\langle c, m, 5, True\rangle$ and $\langle b, n, 7, True\rangle$. $\Diamond$

3.4.3 Top-down Refinement

The top-down refinement stage, detailed in Algorithm 4, aims at improving the matches of the descendants of partially matched target nodes. Differently from the identical subtree matching

Algorithm 3 Refine Bottom-Up

```
1: function refineBottomUp(t,ref matches)
2:     for each d in γ_E(t) do
3:         if d is not leaf and matches[d] = Λ then
4:             refineBottomUp(d, matches)
5:         end if
6:     end for
7:     candidates ← ∅
8:     for each cand in {P(c) | ⟨c,d,w,r⟩ ∈ matches ∧ d ∈ γ_E(t)} do
9:         C_cand ← {⟨c,d,w,r⟩ | ⟨c,d,w,r⟩ ∈ matches ∧ P(c) = cand ∧ d ∈ γ_E(t)}
10:        candidates.add(⟨cand, C_cand, HIPS(C_cand)⟩)
11:    end for
12:    ⟨s, C_s, HC_s⟩ ← candidates.getBest()
13:    matches[t] ← ⟨s, t, ψ(HC_s), False⟩
14:    matches ← matches \ (C_s \ HC_s)
15:    matches ← matches \ {⟨c,d,w,r⟩ | ⟨c,d,w,r⟩ ∈ matches ∧ P(d) = t ∧ P(c) ≠ s}
16: end function
```

stage, we look for source subtrees that are similar or identical, even if they occur more than once in the source document.

Specifically, the top-down refinement works as follows. The target tree is visited top-down and for each visited node t, which is partially matched with a node s, the algorithm tries to improve the matches among the children of s and the children of t. Ideally, we would like to consider the set $M = \{\langle c, d\rangle \mid \langle c, d\rangle \wedge c \in \gamma_E(s) \wedge d \in \gamma_E(t)\}$ of all possible matches among the children of s and those of t and to identify the heaviest consistent subset HM of M, updating the $matches$ set accordingly (that is, if we discard a match among two nodes s and t, we also discard any match among their descendants).

While, as discussed in Section 3.4.2, identifying one of the heaviest consistent subset of a given set of matches is an efficient operation, the size of M, that is, $(|\gamma_E(s)| * |\gamma_E(t)|)$, could be significant. Thus, we consider a smaller set of candidate matches $candM$ which contains, for each child d of t, (i) the match of d in $matches$, if any; (ii) all complete matches between d and a child of s; (iii) the k best matches among d and a subset of the children of s, produced by similarityMatches method, detailed in Algorithm 9, Section 3.6. The Section 3.6, in what follows, is indeed fully devoted to the complex subproblems of reducing the number of comparisons for computing the top-k matches and of estimating the tree edit-distance cost. Once the candidate matches set has been identified, we determine one of its heaviest consistent subsets, updating the $matches$ map accordingly.

A full example of the top-down refinement stage is given in Example 3.6.

Example 3.6. Consider the source and target trees in Figure 3.2 and the detected matches reported in Table 3.1, after the first two stages. In the top-down refinement stage (that is, the third one), when considering node $article_t^2$, its match $\langle article_s^2, article_t^2\rangle$ and the matches of its children $\langle title_s^5, title_t^5\rangle$ and $\langle volume_s^9, volume_t^7\rangle$, the novel matches $\langle author_s^3, author_t^3\rangle$ and $\langle number_s^{11}, number_t^9\rangle$ are established. Similarly, when the node $inproc_t^{22}$, its match $\langle article_s^{13}, inproc_t^{22}\rangle$ and the matches of its children $\langle author_s^{14}, author_t^{23}\rangle$, $\langle title_s^{16}, title_t^{25}\rangle$, and $\langle volume_s^{20},$

Algorithm 4 Refine Top-Down

```
1: function refineTopDown(s, t, origMatches, ref matches)
2:     currM ← {⟨p, q, w, r⟩ | ⟨p, q, w, r⟩ ∈ matches ∧ q ∈ γ_E(t)}
3:     candM ← {⟨p, q, Ω(t), True⟩ | q ∈ γ_E(t) ∧ p ∈ identicalST(q) ∧ P(p) = s} ∪ currM ∪ similarityMatches(s, t, currM)
4:     matches ← matches ∪ HIPS(candM) \ (currM \ HIPS(candM))
5:     for each d in γ_E(t) do
6:         currMatch ← matches[d]
7:         origMatch ← origMatches[d]
8:         if currMatch ≠ ⟨Λ, Λ, Λ, Λ⟩ and !currMatch.c then
9:             if !origMatch.c and origMatch.s ≠ currMatch.s then
10:                matches ← matches \ {⟨p, q, w, r⟩ | ⟨p, q, w, r⟩ ∈ matches ∧ q ∈ D(d)}
11:            end if
12:            if γ_E(currMatch.s) ≠ [] or γ_E(d) ≠ [] then
13:                refineTopDown(currMatch.s, d, origMatches, ref matches)
14:            end if
15:        end if
16:    end for
17: end function
```

Algorithm 5 Edit-script Generation

```
1: function generateEditScript(s, t, matches)
2:     repES ← {repN(s, T(t))}
3:     transES ← compareNode(s, t)
4:     for each unmatched node c in γ_E(s) do
5:         transES ← transES ∪ del(c))
6:     end for
7:     for each partially matched node d in γ_E(t) do
8:         transES ← transES ∪ generateEditScript(matches[d].s, matches[d].t, matches)
9:     end for
10:    for each unmatched node u in γ_E(t) do
11:        transES ← transES ∪ generateInsert(u)
12:    end for
13:    if ξ(repES) ≥ ξ(transES) then
14:        return transES
15:    else
16:        return repES
17:    end if
18: end function
```

$volume_t^{27}\rangle$ are considered, the novel match $\langle number_s^{22}, number_t^{31}\rangle$ is established. Finally, when the node $article_t^{33}$, its match $\langle article_s^{24}, article_t^{33}\rangle$ and the matches of its children $\langle author_s^{25}, author_t^{34}\rangle$, $\langle title_s^{27}, title_t^{36}\rangle$, $\langle year_s^{29}, year_t^{38}\rangle$, and $\langle volume_s^{31}, volume_t^{40}\rangle$ are considered, the new match $\langle number_s^{33}, number_t^{42}\rangle$ is established. $\Diamond$

3.4.4 Edit-script Generation

The last stage of the PUL-Diff algorithm is the edit-script generation. Since matches are propagated towards the root, in the bottom-up refinement stage, the two tree roots are partially matched whenever at least one of their descendant is matched. Algorithm 5 visits the partially matched nodes in a bottom-up fashion and, for each visited pair of partially matched nodes s and t, the algorithm contrasts the cost of replacing $\mathcal{T}(s)$ with $\mathcal{T}(t)$, and the cost of transforming (through

ren, repV, ins$^{\leftarrow}$, repC, ins$^{\searrow}$, and del operations) $\mathcal{T}(s)$ in $\mathcal{T}(t)$, according to the identified matches. The less expensive alternative is then chosen. Intuitively, the cost of the transformation decreases as more and more (valid and consistent) matches are identified among the nodes in $\mathcal{T}(s)$ and those in $\mathcal{T}(t)$.

Specifically the transformation edit-script is generated as follows. First, the value and name of s and t are contrasted by means of the compareNode function, generating a repV or a ren operation, as required. Then, a del (resp. ins$^{\leftarrow}$/ins$^{\searrow}$) operation is generated for every unmatched children of s (resp. t). Insertion operations are produced by means of the generateInsert function. Finally, the algorithm is recursively called on all of the partially matched children of t. Note that, to further reduce the edit-script cost and produce deterministic edit-scripts, we might merge some operations (*e.g.*, whenever adjacent siblings have to be inserted, a single insertion or node replacement is generated). Moreover, repC operations are generated whenever all the children of a source node are replaced with a single text node.

In Example 3.7, the edit-script corresponding to the final set of matches of Table 3.1 is given.

Example 3.7. Consider the source and target trees in Figure 3.2 and the detected matches reported in Table 3.1. The following edit-script is generated:

repV($author^3$,'David Wong'),
del($year^7$),
repV($number^{11}$,'10'),
ins$^{\leftarrow}$($article^{13}$, <article><author>Laurens De Vocht</author>
<title>Title 2.</title><year>2012</year><volume>7</volume>
<number>52</number></article>),
ren($article^{13}$,'inproc'),
ins$^{\leftarrow}$($number^{22}$, <year>2007</year>),
del($year^{18}$),
repV($number^{33}$,'2'),
repN($article^{35}$, <article><author>Massimiliano Pieraccini</author>
<title>Title 5.</title><year>2006</year><volume>44</volume>
<number>11 − 2</number></article>)

In Listings 3.1 we express the same edit-script by means of the equivalent XQuery Update expression. Since a PUL can be produced evaluating the expression, the two representations can be interchanged. ◊

```
replace value of node /dblp/article[1]/author[1] with 'David Wong',
delete node /dblp/article[1]/year,
replace value of node /dblp/article[1]/number with '10',
```

```
insert node (<article><author>Laurens De Vocht</author>
<title>Title 2.</title><year>2012</year><volume>7</volume>
<number>52</number></article>) before /dblp/article[2],
rename node /dblp/article[2] as 'inproc',
insert node (<year>2007</year>) before /dblp/article[2]/number,
delete node /dblp/article[2]/year,
replace value of node /dblp/article[3]/number with '2',
replace node /dblp[1]/article[4] with
(<article><author>Massimiliano Pieraccini</author>
<title>Title 5.</title><year>2006</year><volume>44</volume>
<number>11 − 2</number></article>)
```

Listing 3.1: Edit-script transforming the tree on the left to the tree on the right of Figure 3.2, equivalent to the edit-script of Example 3.7 but expressed using XQuery Update language.

3.5 HIPS: Heaviest Increasing Point Subset

In the bottom-up refinement stage, we want to determine the heaviest consistent subset of a set of matches, where all the source and target nodes share the same parent. We map this problem into the problem of identifying, starting from a set of two-dimensional weighted points M, one of the heaviest subsets of M that defines a strictly increasing function. Specifically, we represent each match $\langle c_i, d_j, w, r\rangle$ in C_{cand}, where i and j denote the index of c_i and d_j in the respective sibling sequences, as a weighted point $\langle i, j, w\rangle$.

Since all matched source nodes and all matched target nodes are siblings in their respective trees, match incompatibility (as originally given in Definition 3.6) can be reformulated as in Definition 3.8.

Definition 3.8. Let $m = \langle i, j, w\rangle$ and $m' = \langle i', j', w'\rangle$ be two matches represented by means of weighted points. We say that m and m' are *compatible* iff either $i > i' \wedge j > j'$ or $i < i' \wedge j < j'$. △

In the remainder of the section, we first present the problem statement along with some basic definitions and notations used in what follows (Section 3.5.1), then we provide an exact $\mathcal{O}(|M| \log |M|)$ algorithm along with the related correctness and complexity proofs (Section 3.5.2).

3.5.1 Problem Statement

Given a sequence σ over a linearly ordered alphabet Σ, finding one of the longest subsequences of σ that is strictly increasing is known as the *Longest Increasing Subsequence* (LIS) problem.

Jacobson and Vo [JV92] generalize the LIS problem to weighted sequences, defining the *Heaviest Increasing Subsequence* (HIS) problem. Given a sequence σ over Σ and a weight function defined on the symbols and their positions in σ, the HIS problem is to find one of the subsequences of σ with the highest sum of weights, among those linearly increasing. The algorithm they propose is optimal and has time complexity in $\mathcal{O}(n \log n)$.

Let M be a set of matches, represented as points. Assuming that the x component of each match is unique, we can determine the heaviest consistent subset of M by applying the HIS algorithm on the sequence consisting of the y components of the matches in M, sorted in increasing order of their x component.

Since in our setting multiple matches can be present for the same source node, in this section we tackle a generalization of the HIS problem, the *Heaviest Increasing Point Subset* (HIPS), where, for each position in the sequence, multiple mutually exclusive alternatives may be present.

For sake of clarity, we use the point-based notation, where a point p is a triple $\langle x, y, w \rangle$, where the x and y component of each point are chosen from two linearly ordered alphabet X and Y, respectively, and w denotes the point weight. We use $p.x$, $p.y$, and $p.w$ to denote the point components. The weight of a set of points M, denoted as $\psi(M)$, consists of the sum of the weights of its elements. Specifically, given a set M of points, the HIPS problem is to identify one of the heaviest subsets of M that defines a strictly increasing function from X to Y. More formally, a point set M is *increasing* if and only if, for each pair of distinct point p_1 and p_2 in M we have that $(p_1.x > p_2.x \iff p_1.y > p_2.y) \wedge (p_1.x < p_2.x \iff p_1.y < p_2.y)$ hold.

A more formal definition of the HIPS problem is given in Definition 3.9.

Definition 3.9. Let M be a point set. An increasing subset H of M is a *heaviest increasing point subset* of M if no other increasing subset of M has a higher weight. $\triangle$

Let $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n$ be a point sequence. Following the conventions proposed in [JV92], we denote by σ_i the i-th element of σ. The weight of σ is the sum of its components weights, denoted as $\psi(\sigma)$.

We say that a point sequence $\tau = \tau_1 \tau_2 \ldots \tau_l$ is a *subsequence* of σ if a sequence of integers $i_1 < i_2 < \ldots < i_l$ exists such that τ is equal to $\sigma_{i_1} \sigma_{i_2} \ldots \sigma_{i_l}$. We denote by $\sigma_{i \ldots j}$ the contiguous subsequence of σ consisting of the points from position i to position j. We say that τ is *increasing* if $\tau_1.x < \tau_2.x < \ldots < \tau_l.x$ and $\tau_1.y < \tau_2.y < \ldots < \tau_l.y$ hold.

In Definition 3.10, the defininition of a heaviest increasing point subsequence is given.

Definition 3.10. Given a point sequence σ, an increasing point subsequence τ of σ is a heaviest increasing point subsequence if no other increasing point subsequence of σ has a higher weight. $\triangle$

Note that the heaviest increasing point subsequence may not be unique (that is, different increasing subsequences with the same weight may exist), as shown in Example 3.8.

Example 3.8. Consider the point sequence $\sigma = \langle 1,2,1\rangle\ \langle 2,3,1\rangle\ \langle 2,4,1\rangle\ \langle 3,5,1\rangle$ and the following two point subsequences of σ: $\tau = \langle 1,2,1\rangle\langle 2,3,1\rangle\langle 3,5,1\rangle$ and $\upsilon = \langle 1,2,1\rangle\langle 2,4,1\rangle\ \langle 3,5,1\rangle$. According to Definition 3.10, both τ and υ are heaviest increasing point subsequences of σ. $\Diamond$

In the algorithm we need to compare increasing subsequences of σ and to decide whether they can be a subsequence of a heaviest increasing point subsequence of σ or not. For this reason, we introduce the notion of dominated point subsequence in Definition 3.11.

Definition 3.11. Consider two increasing point subequences of a point sequence σ, $\tau = \tau_1\tau_2\ldots\tau_m$ and $\upsilon = \upsilon_1\upsilon_2\ldots\upsilon_n$, and let ψ_τ and ψ_υ be the two sequence weights. We say that υ is *dominated* by τ iff $\upsilon_n.y \geq \tau_m.y \wedge \psi_\tau > \psi_\upsilon$ or $\upsilon_n.y > \tau_m.y \wedge \psi_\tau \geq \psi_\upsilon$. $\triangle$

An example of a dominated subsequence is given in Example 3.9.

Example 3.9. Consider the point sequence $\sigma = \langle 1,2,1\rangle\ \langle 2,2,1\rangle\ \langle 2,3,1\rangle\ \langle 3,4,1\rangle$ and the two following point subsequences of σ: $\tau = \langle 1,2,1\rangle\langle 2,3,1\rangle$ and $\upsilon = \langle 2,3,1\rangle\langle 3,4,1\rangle$. According to Definition 3.11, υ is dominated by τ. $\Diamond$

3.5.2 Algorithm Description

Algorithm 6 HIPS Algorithm

```
1: function HIPS(M)
2:     σ = σ_1σ_2...σ_n ← the sequence of points in M sorted in ascending order on the point x component first, on y component second
3:     S ← ∅
4:     P ← ∅
5:     SW ← ∅
6:     AQ ← []
7:     currXMaxW ← −1
8:     for each i in [1...n] do
9:         pred ← S.pred(⟨0, σ_i.y, σ_i.w⟩)
10:        SW[σ_i] ← SW[pred] + σ_i.w
11:        P[σ_i] ← pred
12:        if SW[σ_i] > currXMaxW then
13:            currXMaxW ← SW[σ_i]
14:            succ ← S.succ(pred)
15:            if succ = Λ or (σ_i.y < succ.y or SW[succ] < SW[σ_i]) then
16:                AQ.addLast(σ_i)
17:            end if
18:        end if
19:        if i = n or σ_i.x ≠ σ_{i+1}.x then
20:            processQueue(ref S, P, SW, AQ)                ▷ Shown in Algorithm 7
21:            currXMaxW ← −1
22:            AQ.clear()
23:        end if
24:    end for
25:    return maximalSubset(S, P)                            ▷ Shown in Algorithm 8
26: end function
```

Algorithm 7 HIPS Algorithm (processQueue function)

```
1: function processQueue(ref S, P, SW, AQ)
2:     for each μ in AQ do
3:         succ ← S.succ(P[μ])
4:         while succ ≠ Λ do
5:             if SW[μ] < SW[succ] then
6:                 break
7:             end if
8:             S.remove(succ)
9:             succ ← succ(succ)
10:        end while
11:    end for
12:    for each μ in AQ do
13:        S.add(μ)
14:    end for
15: end function
```

Algorithm 8 HIPS Algorithm (maximalSubset function)

```
1: function maximalSubset(S, P)
2:     MS ← ∅
3:     curr ← S.max()
4:     while curr.x ≠ τ do
5:         MS ← MS ∪ {curr}
6:         curr ← P[curr]
7:     end while
8:     return MS
9: end function
```

Given the specific context in which we employ the algorithm, without loss of generality we make the following assumptions on the input data. Both X and Y alphabets are $\mathbb{N}_0$, each symbol pair represents a point in a two-dimensional space, all points are different, all weights are greater or equal to 0.

Intuitively, the algorithm considers a set of points M and sorts it in ascending order on the point x component first, and then on the point y component, producing a point sequence $\sigma = \sigma_1 ... \sigma_n$.

Let $Y_{1...i} = \{p.y \mid p \in \sigma_{1...i}\}$ be the set of all the y components of the points in $\sigma_{1...i}$. The algorithm scans σ (from the first to the last element) and keeps updated an auxiliary set S. After σ_i symbol has been considered, the set S represents one of the non-dominated subsequences of σ whose last point y component is $p.y$, for each $p.y \in Y_{1...i}$ (if at least one exists). When the last symbol of σ has been considered, the highest-weight subsequence represented in S (the one with the highest y component) contains all and only the points in one of the heaviest increasing point subsets of M.

Specifically, the point set S is kept sorted in ascending order on the points y component first, and on the x component second, and provides the following functions (relying on the ordering of S):

(i) pred(p), which, given a point p, returns the point p' in S such that p' is the majorant of the minorants of p. If no such point exists, the function returns Λ.

(i) succ(p), which, given a point p, returns the point p' in S such that p' is the minorant of the majorants of p. If no such point exists, the function returns Λ. For brevity we denote succ(Λ) as min().

(i) max() returns the greatest element in S, if S is not empty, Λ otherwise.

(i) min() returns the least element in S, if S is not empty, Λ otherwise.

More precisely, the algorithm represents non dominated subsequences of σ by means of the point set S and two additional maps, SW and P. For each subsequence τ, S stores the last considered element of τ, SW stores the subsequence weight, while P associates each point in τ with its predecessor (allowing us to reconsctruct the actual subsequence through a backward traversal).

Algorithm 6 starts by sorting the input set of points M in ascending order on the point x component first, and on the point y component then, obtaining a point sequence σ (line 2).

Then, for each point σ_i in σ, the algorithm determines the heaviest subsequence τ (which is terminated by the $pred$ point identified at line 9) in S to which σ_i can be appended. When the, possibly empty, sequence τ has been determined, the algorithm updates the SW and P maps accordingly (lines 10–11). If $\tau\sigma_i$ is not dominated by another sequence ending at an element with the same x component of σ_i, it is scheduled for addition to S (line 16).

When all the points with a given x component are visited, those scheduled for addition, as well as the point set S and the maps SW and P, are given as input to the processQueue function (lines 19–23). processQueue function first removes from S (lines 2–11, Algorithm 7) any subsequence which is dominated by one of those scheduled for addition, then adds the scheduled subsequences to S (lines 12–14 Algorithm 7).

Finally, when all the input points have been processed, the heaviest increasing point subset of M is returned using the function maximalSubset (Algorithm 8), that builds the heaviest increasing subsequence with a traversal of the predecessor map P, starting from the maximal point $S.max()$.

The correctness of the HIPS algorithm is stated in Proposition 3.1, along with a proof sketch.

Proposition 3.1 (Correctness of Algorithm 6)**.** Given a set of points M as input, Algorithm 6 returns one of the heaviest increasing subsets of M. □

Sketch Proof for Proposition 3.1. Assume that the input point set contains no two points with the same x component and let $Y_{1...i} = \{p.y \mid p \in \sigma_{1...i}\}$ be the set of all y components of the points in $\sigma_{1...i}$. Under this assumption the proposition can be proved by induction by observing that, after the i-th element of the input has been processed, the set S represents one of the non-dominated subsequences (if any) whose last point y component is $p.y$, for each $p.y \in Y_{1...i}$.

Base case: After the first element σ_1 has been considered, $Y_{1...1} = \{\sigma_1.y\}$, and the algorithm represents the only non-dominated increasing point subsequence of $\sigma_{1...1}$ whose last point y component is $\sigma_1.y$, that is, $[\sigma_1]$ in S.

Inductive case: Assume now that, after the i-th element of σ has been considered, S represents one of the non-dominated subsequences (if any) whose last point y component is $p.y$, for each $p.y \in Y_{1...i}$.

When the point σ_{i+1} is considered, the algorithm ensures that at least one non-dominated increasing subsequence of σ_{i+1}, ending with a point whose y component is $\sigma_{i+1}.y$, is represented in S.

Due to the inductive hypothesis, we can observe that either one of the highest weight subsequences of $\sigma_{1..i+1}$ is already represented in S or it ends with σ_{i+1}.

In order to determine one of the highest weight subsequences of $\sigma_{1..i+1}$ ending with σ_{i+1}, we rely on the following observation. For any two non dominated point subsequences $\tau = \tau_1\tau_2 \dots \tau_l$ and $\upsilon = \upsilon_1\upsilon_2 \dots \upsilon_m$ of σ, where $\tau_l.y \neq \upsilon_m.y$, according to Definition 3.11, either $\tau_l.y > \upsilon_m.y \wedge \psi(\tau) > \psi(\upsilon)$ or $\tau_l.y < \upsilon_m.y \wedge \psi(\tau) < \psi(\upsilon)$.

From this observation and the inductive hypothesis, it follows that one of the highest weight subequences ending with σ_{i+1} consists of the subsequence represented in S ending with the point with the highest y component smaller than $\sigma_{i+1}.y$ (if such an element exists in S, [] otherwise) concatenated with σ_{i+1}. This sequence, if not dominated, is represented in S by means of the processQueue function (Algorithm 7).

Therefore, after the last point of σ has been considered, the sequence represented in S, ending with the point with the highest y component, is one of the heaviest increasing point subsequence of σ.

Since σ is sorted on the points x component in ascending order first, we can easily observe that each heaviest increasing point sequence of σ contains all and only the points of one of the heaviest increasing point subset of M.

The proof can then be easily extended to arbitrary input sequences, where multiple points with the same x component may be present, observing that the algorithm considers increasing sequences consisting of points with different x components, thus proving the proposition.

After the detailed description of the HIPS algorithm, and its subcomponents, we are now ready to provide an analysis of its time complexity, along with a proof sketch (Proposition 3.2).

Proposition 3.2. Given a sequence $\sigma = \sigma_1\sigma_2 \dots \sigma_n$, the time complexity of the HIPS algorithm is in $\mathcal{O}(n \log n)$. □

Sketch Proof for Proposition 3.2. In Algorithm 6, at most n invocations of the pred and succ functions are required by the HIPS function. The processQueue function (Algorithm 7), which

is invoked at most n times, requires across all invocations to evaluate n times the succ function (specified at line 3). The succ function (line 9) is also evaluated at most n times since it is only invoked when an element is removed from S, which contains at most n elements. If S is efficiently implemented, for instance using self-balancing binary search trees, both pred and succ can be evaluated in $\mathcal{O}(\log n)$ time, thus proving the proposition.

3.6 Approximate Tree Matching

In the top down-refinement stage, we consider partial matches and try to improve them, that is, our objective is to increase the weight of the matches among the children of the two partially matched nodes. Let $\langle s, t, w, False \rangle$ be one of the considered matches. Ideally, we would like to identify one of the heaviest consistent subset of all possible matches among the children of s and those of t. While identifying one of the heaviest consistent subset of a given set of matches is an efficient operation, the number of all the possible matches could be large. Moreover, exactly determining the minimum cost of a PUL transforming a tree into another is extremely expensive.

In the remainder of the section, we tackle both problems. Specifically, in Section 3.6.1 we introduce our technique based on *pq*-gram for estimating the minimal cost of a PUL transforming a tree into another, whereas in Section 3.6.2 we present an approach for choosing candidate matches between two tree sequences.

3.6.1 Tree PUL Edit-distance

One of the most relevant problem for tree data structures is the tree edit-distance, that is, to determine, given two trees S and T, a set of edit operations, and an associated cost function, the minimum cost of a sequence of edit-operations which transform S into T.

Exact algorithms, such as those in [DMRW09, ZS89], have optimal time complexity in $\mathcal{O}(n^3)$. To estimate PUL edit-distance, we extend the *pq*-gram approximated similarity measure introduced in [ABG10].

pq-grams are a generalization of string *q*-grams: they are subtrees of fixed size and shape, composed by a *stem* made of p elements (bound by the parent-child relation) and a *base* of q consecutive siblings: the last element of the stem (starting from the root node) is named *anchor node*, and the element of the base are children of this node.

To ensure that each node of the tree appears in at least one *pq*-gram as an anchor node, the tree is extended with dummy nodes not occurring in the original tree. Moreover, the textual content of element nodes is concatenated to their name, and text nodes are removed. The obtained tree is named *extended tree representation*. The similarity between two trees can then be estimated

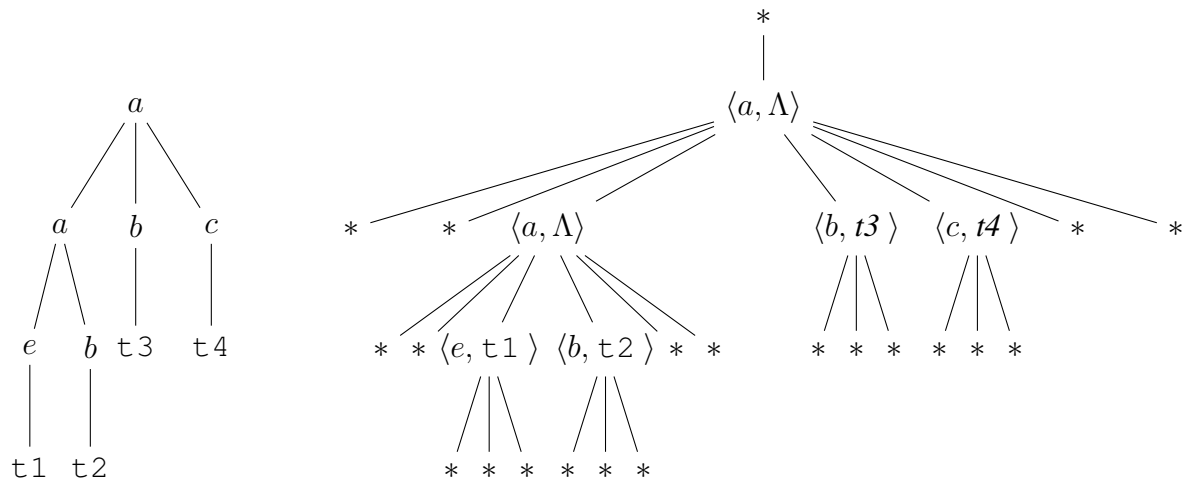


Figure 3.4: Original tree t (left) and the corresponding extended tree t' (right).

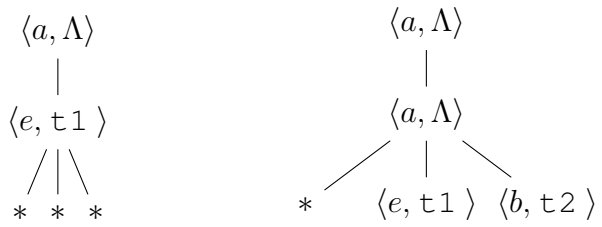


Figure 3.5: Two *pq*-grams for tree t of Figure 3.6.

computing the Jaccard-distance of the sets of all *pq*-grams extracted from the two trees extended representations.

In our context, since "move" operations are not allowed, identical *pq*-grams whose anchor nodes are at different nesting levels should not increase the similarity of the two trees under analysis. Moreover, we experimentally observed that *pq*-grams tend to under-estimate the similarity of trees where nodes have different labels or values with respect to our PUL cost model, because they embed node labels and values in the same *pq*-gram.

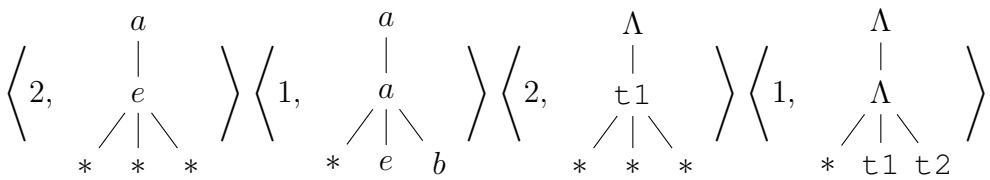


Figure 3.6: Two name-*pql*-grams (left) and two value-*pql*-grams (right) for tree t of Figure 3.6.

To overcome these limitations, we introduce *pql*-grams, in two flavors: name *pql*-grams and value *pql*-grams. Both are generated from the same extended representation used for generating *pq*-grams, but name *pql*-grams contain only node names, whereas value *pql*-grams contain only node values. Moreover, both kinds of *pql*-grams include the nesting level of their anchor nodes. Tree similarity is computed as for *pq*-grams.

Consider two trees S and T. Starting from the similarity measure α, based on *pql*-gram, between them (which ranges from 0, when two trees share no *pql*-gram, to 1, when they have the same *pql*-grams), we estimate the PUL edit-distance as the quantity $(1 - \alpha) * (\Omega(S) + \Omega(T))/2$. An experimental evaluation of the quality of the PUL edit-script distance estimation based on *pql*-gram can be found in Section 3.7.2.

In Example 3.10 a tree is given, along with the associated extended representation, and some of its *pq*-grams and *pql*-grams.

Example 3.10. Figure 3.4 presents a tree t (left) and its extended representation t' (right), while Figure 3.5 shows an example of two *pq*-grams from t'. In Figure 3.5, instead, two name and value *pql*-grams with the same anchor nodes are given, again computed starting from t'. The first component of *pql*-grams represents the nesting level of the anchor node. $\Diamond$

3.6.2 Window-based Sequence Matching

In the top-down refinement stage (see Section 3.4.3), we contrast the sequence of children of two partially matched nodes with the aim of identifying the heaviest consistent subset of all the possible matches among nodes in the two sequences.

In this section, we propose a window-based approximated approach, which uses *pql*-grams to estimate the tree PUL edit-distance between two trees. The motivation behind this approaximated approach is twofold: first, the exact solution would require to produce a similarity matrix among all the elements of the two sequences, second, for each of the element of the matrix, a computation of tree edit-script distance would be required. As already stated in the introduction, the tree edit-script distance computation of is an extremely expensive task, and the resulting approach would therefore be impractical.

In our proposal we start by defining a search window for each unmatched node of the target sequence, exploiting existing matches among nodes in the two sequences. We partition the source and target sequences in consecutive partitions, each delimited by two currently matched nodes (first/last are delimited by the start/end of the node sequence). Since the current matches are consistent, we expect the best match for the nodes in the i-th target sequence to lie in the i-th source partition. More precisely, we consider the probability that the best match for a node of the i-th sequence can be found in the j-th source sequence is inversely proportional to the quantity $|i - j|$.

While using larger windows grants an higher probability of finding the best match for a given target node, it also increases the probability that the considered matches are incompatible with the other matches. For this reason, using large windows (or even the entire source sequence) do not usually increase the quality of the identified matches, as experimentally verified in Section 3.7.2.

In Example 3.11 we provide a toy example in order to clarify some of the main aspects of the proposed approach.

Example 3.11. Consider a partial match $\langle s, t, 6, False \rangle$ and let the node sequence reported at the top (resp. bottom) of Figure 3.7 to be the sequence of children of s (resp. t), where matches are depicted with a line. A simple heuristic for defining a search window for each target node d is to scan in both directions the target children sequence, starting from d, and to consider the window defined by the two source nodes matched by the first encountered matched node in each direction. If the start (resp. end) of the target sequence is reached, the start (resp. end) of the source children sequence is used instead. For instance, using this criteria, the search window for K is $[C..G]$, whereas for O it is $[G..H]$. ◊

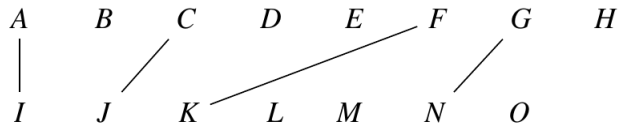


Figure 3.7: Source sequence (top) and target sequence (bottom).

As already introduced, a further optimization is to avoid a 1-to-1 comparison between a target node and each node in its source window. Rather, we compare all target nodes with these source window subtrees at the same time. Specifically, let $d_1, \ldots, d_n$ and $c_1, \ldots, c_m$ be a set of target nodes and their common source window, respectively. We first compute and sort the sets P_s (resp. P_t) of all source (resp. target) window *pql*-grams and we tag each *pql*-gram with the subtree of the source (resp. target) window it belongs to. Then, through a scan of both sets we populate an $m \times n$ matrix M, where the $M_{i,j}$ cell contains the number of *pql*-grams in common between c_i and d_j. Starting from this matrix, we can easily compute the *pql*-gram distance. We remark that, even if the worst case complexity does not change, this approach is more efficient than 1-to-1 comparisons and yields a speedup proportional to the number of unique *pql*-grams.

The proposed method is descibed in Algorithm 9. The method accepts two partially matched nodes s and t, and first partitions the sequences of children of s and t, so that matched nodes occur only as the first/last element of a partition (lines 3–4). Let n be the number of identified partitions. The algorithm then estimates the similarity between all the subtrees in the p-th source and target partition (lines 5–12). To compare two partitions the algorithm first computes, by means of function pqlGrams (lines 6–7), for each node c (resp. d) in the source (resp. target) partition, the set of all *pql*-grams of $\mathcal{T}(c)$ (resp. $\mathcal{T}(d)$), along with their number of repetitions in the set of *pql*-grams of $\mathcal{T}(c)$, and the node c (resp. d). More precisely, this information is

Algorithm 9 Similarity Matches

```
1: function similarityMatches(s, t, matches)
2:     candMatches ← ∅
3:     {s_1, ..., s_n} ← the partitions of γ_E(s) according to the matched nodes
4:     {t_1, ..., t_n} ← the partitions of γ_E(t) according to the matched nodes
5:     for each p in [1...n] do
6:         sGrams ← {⟨pql, o, c⟩ | ⟨pql, o⟩ ∈ pqlGrams(c) ∧ c ∈ s_p}
7:         tGrams ← {⟨pql, o, d⟩ | ⟨pql, o⟩ ∈ pqlGrams(d) ∧ d ∈ t_p}
8:         M ← a zero-filled |s_p| × |t_p| matrix
9:         M[i][j] ← Σ_{⟨pql,o1,o2⟩∈X} min(o1, o2)
10:            where X = {⟨pql, o1, o2⟩ | ⟨pql, o1, c_i⟩ ∈ sGrams ∧ ⟨pql, o2, d_j⟩ ∈ tGrams}
11:        candMatches.add(heaviestMatches(M, s, t))
12:    end for
13:    return candMatches
14: end function
```

encoded as the triple $\langle pql, o, c\rangle$, where o is the number of repetitions of the *pql*-gram pql in the tree rooted at node c.

The algorithm then populates the matrix M (lines 8–10), as described in the previous paragraph, and contains, in the end, the number of *pql*-grams in common between each pair of source and target subtrees. The k-heaviest matches for each target window node are then identified and added to the candidate matches map, by means of function heaviestMatches (line 11). When all the partitions have been processed, all the candidate matches are returned.

We stress that, in the PUL-Diff algorithm, the search window for a given target node d depends on $\Omega(d)$, and on the weights of the matches delimiting the subsequences adjacent to the one d belongs to. Additionally, we enforce an upper-bound for the number of subtrees in the target window to limit the comparison complexity.

After the detailed description of the algorithm, we are now ready to state, in Proposition 3.3, its temporal and spatial complexities, along with the proof sketch for the proposition.

Proposition 3.3 (Complexity of Algorithm 9)**.** Let P and Q be two trees. The time complexity of Algorithm 9, applied on P and Q, is in $\mathcal{O}(n \log n)$, and the spatial complexity is in $\mathcal{O}(n)$, where $n = |\mathcal{V}(P)| + |\mathcal{V}(Q)|$. □

Sketch Proof for Proposition 3.3. As a trivial extension of the proof given in [ABG10], the computation of the *pql*-grams distance between two trees is in $\mathcal{O}(n \log n)$, the same as for *pq*-grams. The number of nodes in both the source and target window is limited to a constant number. For what concerns spatial complexity, the number of *pq*-grams for a tree t is linear in the number of nodes of t [ABG05]. For the same tree, the number of *pql*-grams is twice the number of *pq*-grams, so again linear: *pql*-grams are only used in the window-search approach, whose worst-case requires the computation of the *pql*-grams for both the source and target tree. The worst-case space complexity, for *pql*-grams computation, is therefore in $\mathcal{O}(n)$.

3.7 Experimental Evaluation

In this section we first investigate the PUL-Diff complexity in Section 3.7.1. Then, we present an experimental evaluation of the PUL edit-distance estimation and provide a time and edit-script cost comparison with other state of the art XML differencing algorithms in Section 3.7.2.

3.7.1 Algorithm Complexity

Proposition 3.4 states the temporal and spatial complexity of the PUL-Diff algorithm.

Proposition 3.4. Let S and T be two XML documents. The temporal complexity of PUL-Diff on S and T is in $\mathcal{O}(n \log n)$, whereas the spatial complexity is in $\mathcal{O}(n)$, where $n = |\mathcal{V}(S)| + |\mathcal{V}(T)|$. □

Sketch Proof for Proposition 3.4. The identical subtree matching stage (Algorithm 2) computes a hash signature of each subtree in S and T, using a bottom-up DOMHASH-like approach, with a cost linear in the number of nodes of S and T. Moreover, n hash-table insertions and up to $|\mathcal{V}(T)|$ hash-table lookups must be performed. The space required to store the node hash signatures is in $\mathcal{O}(n)$.

The bottom-up refinement stage (Algorithm 3) performs, for each node in $|\mathcal{V}(T)|$, a (very small) constant number of hash-table insertions/lookups and deletions. Moreover, across all invocations, the HIPS algorithm processes at most $|\mathcal{V}(T)|$ matches. Let M be a set of matches. Since the time complexity of the HIPS algorithm is in $\mathcal{O}(|M| \log |M|)$ (see Proposition 3.2), the worst-case complexity for the second stage is $\mathcal{O}(n \log n)$.

The top-down refinement stage (Algorithm 4) performs, for each node in $|\mathcal{V}(T)|$, a (very-small) constant number of hash-table insertions/lookups and deletions. For reducing the complexity of this stage, the number of identical matches, identified at line 3 by identicalST function, is limited using a window-based approach. Therefore, across all invocations, the window-match algorithm (Algorithm 9) is invoked on at most $|\mathcal{V}(S)|$ source nodes and on at most $|\mathcal{V}(T)|$ target nodes. If we assume a costant tree depth, according to Proposition 3.3, the worst-case time complexity for the third stage is in $\mathcal{O}(n \log n)$. Moreover, the number of matches is always linear in $|\mathcal{V}(T)|$ during this stage, thus the spatial complexity of storing the matches is in $\mathcal{O}(n)$. For what concerns *pql*-grams, according to Proposition 3.3, the spatial complexity is again in $\mathcal{O}(n)$.

The edit-script generation (Algorithm 5) visits both the source and target trees and performs a small constant number of hash-set lookups, for each target node. Moreover, no more than n operations can be generated and no more than $|\mathcal{V}(T)|$ nodes can be inserted. Since the number of generated operation is at most n, the spatial complexity for storing them is again in $\mathcal{O}(n)$.

We can therefore conclude that the temporal complexity of PUL-Diff algorithm is in $\mathcal{O}(n \log n)$, and its spatial complexity is in $\mathcal{O}(n)$, thus proving the proposition.

3.7.2 Experiments

This section provides an experimental evaluation of different aspects of interest of our proposal.

Since no suitable versioned XML document collection is available, we relied on synthetic documents, generated as follows. The source document is either produced by means of the XMark document generator,[4] or selecting a random subset of the first-level children of the DBLP XML document.[5] These two kinds of source documents presents different characteristics. XMark documents have a complex organization, with subtrees representing several different entities, whereas DBLP-based documents are a simple list of subtrees describing scientific publications.

Instead of computing the exact edit-script cost between the source and target documents, we randomly generate a minimal PUL with a known cost (using a complex blocking strategy for the target elements of the generated operations), and we obtain the target document by applying this PUL to the source document.

These PULs contain randomly generated `del`, `ins`^{d}, `repN`, `ren`, `repV` operations. Inserted and replaced subtrees can be randomly generated, similar or equal to another subtree of the source document, or randomly selected from another XMark/DBLP document. New names and values can be randomly generated, or be randomly selected from another XMark/DBLP document. We also simulate subtree moves through pairs of deletion and insertion. The number of each operation type is roughly the same. Although we ran the experiments with many different distributions of operation types, for brevity we only report the results obtained with this distribution. No significant differences were found using XMark and DBLP-based documents.

To easily combine and contrast the results obtained considering documents of different sizes, we do not reason directly on the tree edit-distance, rather we consider the *change ratio*, that is, the edit-distance divided by the source document weight.

In the experiments we considered $p = 2$, $q = 3$ in *pq*-grams and *pql*-grams generations, $k = 5$ for the considered top-k matching candidates, with no limits on the approximated matching window size. The tests has been performed on a PC equipped with an *Intel Core i7-2670QM* CPU, 16GB of RAM and *Kubuntu Linux 12.10 64-bit* O.S. For increasing statistical significancy, every time measurement is averaged over at least 50 samples.

In the remainder of the section, we first experimentally evaluate the quality of the PUL edit-distance estimation obtained by means of tree-grams, then we empirically test the soundness of

[4] `http://www.xml-benchmark.org`

[5] `http://www.informatik.uni-trier.de/~ley/db`

our window-based approach used in top-down refinement stage, and we finally provide a time and edit-script cost comparison with other state of the art XML differencing algorithms.

PUL Edit-distance Estimation

This section experimentally evaluates the quality of the PUL edit-script distance estimation using tree-grams (*i.e.*, *pq*-grams and *pql*-grams). Let S, T, and T' be three trees and let e_1 (resp. e_2) be the edit-script distance between S and T (resp. S and T'). Our goal is to maximize the confidence that if $e_1 > e_2$, then the edit-script distance between S and T is higher than the one between S and T'. Thus, we need that, independently from the kind of operations which are needed to transform a tree into another:

(i) the estimated change ratio increases when the real one does and the increments are similar,

(i) for any given real change ratio, the estimated change ratio has a low variance.

As discussed in Section 3.6, tree-grams distance is only used to select, in a sequence of trees, the k most similar trees to a given one. Even if the k best matches, according to the estimation, are sub-optimal, the application of the HIPS algorithm often chooses a perfect or near-perfect set of matches, as experimentally verified in Section 3.7.2.

We considered both homogeneous and non-homogeneous random PULs, composed by `del`, `ins`^{d}, `repN`, `ren`, `repV` or move operations. When investigating homogeneous PULs, we also experimented with the following additional variables in the generation of the operations:

(i) the deleted subtrees weight range, for `del` operations,

(i) the number of inserted subtrees, their weight range, and generation algorithm for `ins`^{d} and `repN` operations,

(i) the number of adjacent sibling moved subtrees, for move operations,

(i) the name generation algorithm for `ren` PULs,

(i) the value generation algorithm for `repV` PULs.

We considered the following inserted tree generation algorithms:

(i) *random trees*,

(i) trees which have a *similar structure* (10% of the nodes are renamed, 70% of the text values are changed) to either one of the inserted subtree sibling or to a subtree at the same nesting depth,

(i) trees which have a *similar size* to that of their new sibling/replaced node, but random structure, names, and values,

(i) *real trees*, randomly extracted from another XMark/DBLP document, among the subtrees which usually occur in the considered document, at the considered position (*e.g.*, a new `article`/`person` subtree can be inserted next to another `article`/`person` subtree).

Random tree height is limited to 6, and the structure roughly resembles that of an XMark document.

Figures 3.8, 3.9, 3.10 and 3.11 contrast the real and estimated change ratio for PULs with different characteristics, among those which we consider more common in XML databases. Each measurement is repeated 300 times on XMark and DBLP random documents, whose size ranges from 1KB to 2MB. For sake of conciseness we do not report the results for every investigated PUL kind in these figures. The interested reader can refer to Appendix A for the full evaluation results on the different PUL primitives.

For the tests we generated roughly 1 million source documents and PULs, with different change ratios, uniformly distributed among the following values (0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 and 0.7) and the following kinds (random, homogeneous `del`, `ins`^{d}, `repN`, `ren`, `repV` and move). Moreover, for each kind, the PUL are uniformly distributed among the different parameters of interest.

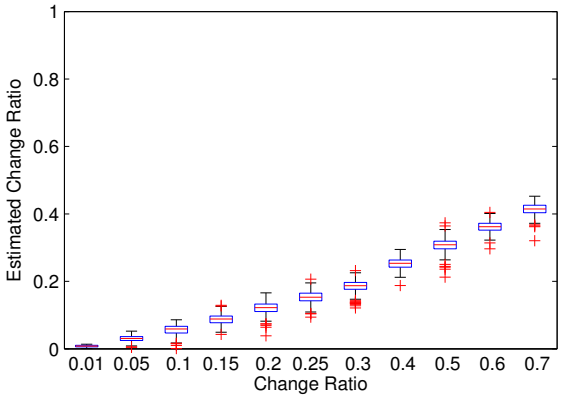
For brevity, in Table 3.2, we only report the results for random non-homogeneous PULs. In these tables we report both the average estimated value and its standard deviation.

We can observe, from the aforementioned tables and figures, that as the real change ratio increases, both the *pq*-gram and the *pql*-gram estimated change ratios increase of a similar amount. *pq*-grams overestimate the edit-script cost when renamings or value-changes are necessary and have a much higher variance. Therefore, at least in our context, *pql*-grams provide a better estimation than *pq*-grams for PUL edit-distance.

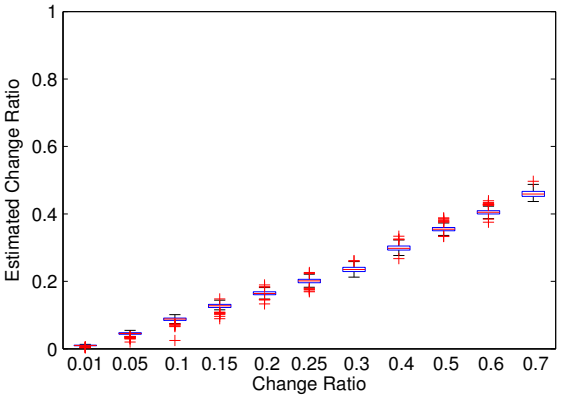
Window-based Sequence Test

In the top-down refinement stage a sequence of source subtrees is contrasted with a sequence of target subtrees, aiming at finding the heaviest consistent set of matches among them. The algorithm employs an approximated approach: each target subtree is compared employing *pql*-grams with all the source subtrees in its search window. The best k matches for each target subtree are inserted into a set M. The HIPS algorithm is then invoked on M identifying the heaviest consistent subset of M.

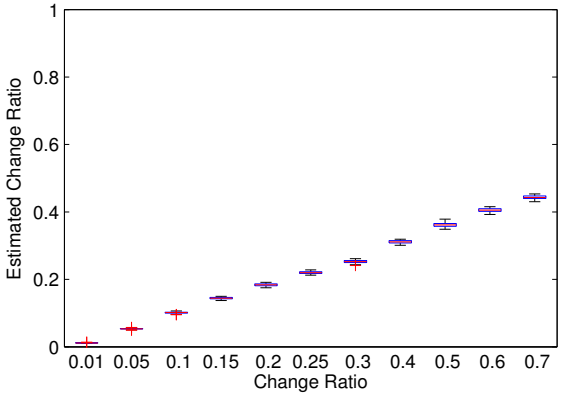
In this experiment, we aim at replicating the window-based match settings producing subtree sequences, with a subtree size ranging from 20 to 60 nodes. Target documents are generated using



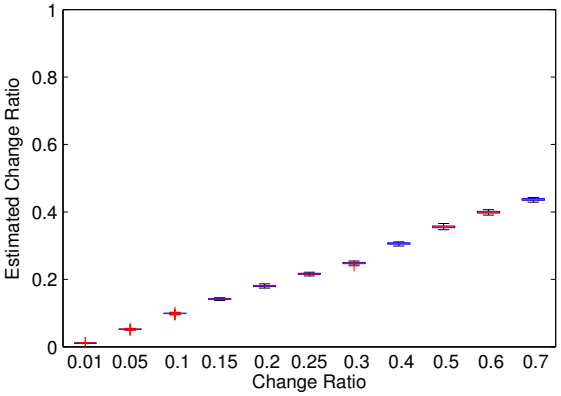
(a) Moves (1-3 siblings, total weight 1-500) (*pq*-grams).



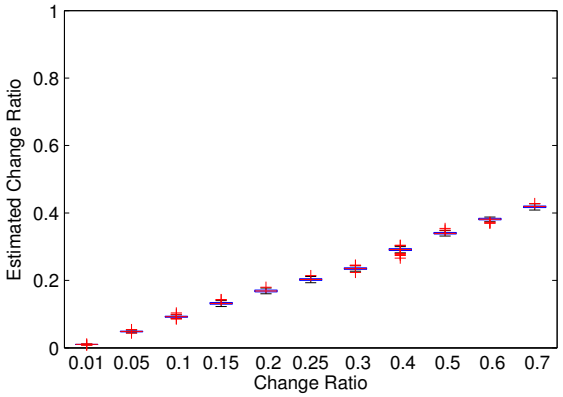
(b) Moves (1-3 siblings, total weight 1-500) (*pql*-grams).



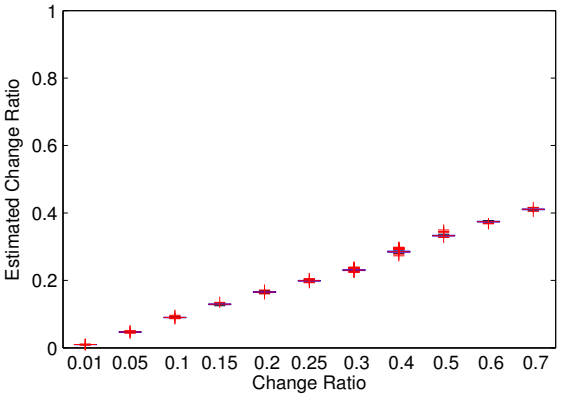
(c) Insertions (1-3 random trees, total weight 1-200) (*pq*-grams).



(d) Insertions (1-3 random trees, total weight 1-200) (*pql*-grams).

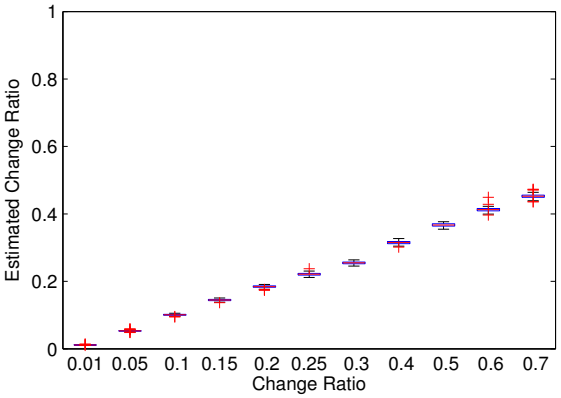


(e) Insertions (1-3 real trees, total weight 1-200) (*pq*-grams).

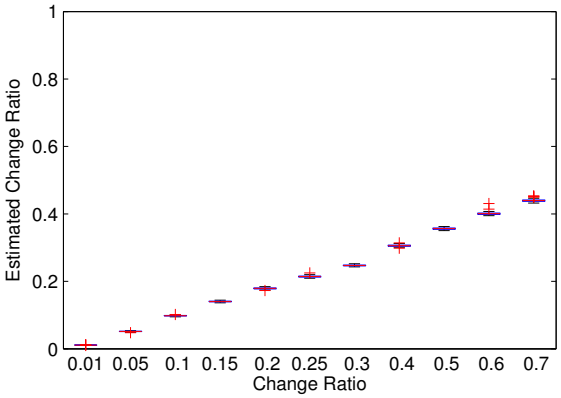


(f) Insertions (1-3 real trees, total weight 1-200) (*pql*-grams).

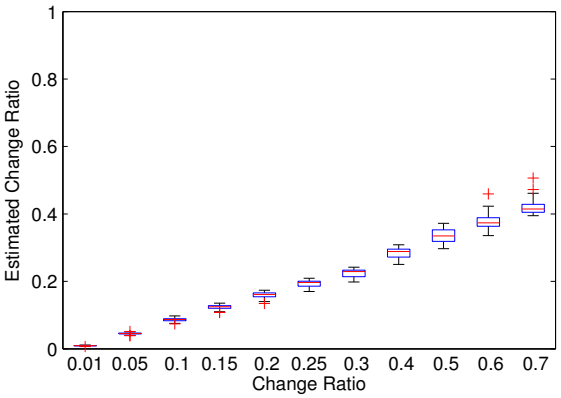
Figure 3.8: Tree PUL edit-distance estimation with *pq*-grams and *pql*-grams (part 1).



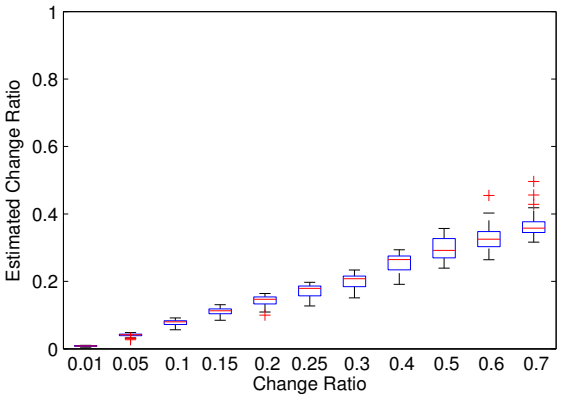
(a) Replacements (of nodes weighting 1-500, with 1-3 random trees with total weight 1-200) (*pq*-grams).



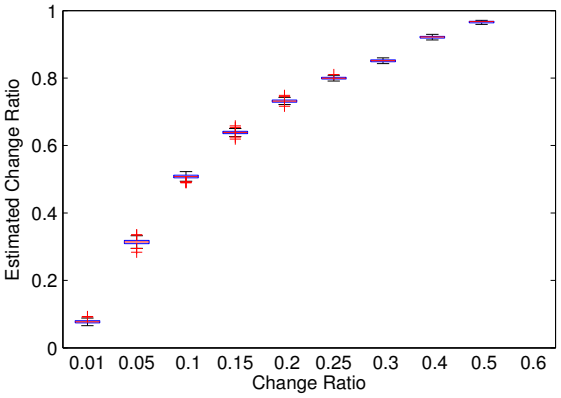
(b) Replacements (of nodes weighting 1-500, with 1-3 random trees with total weight 1-200) (*pql*-grams).



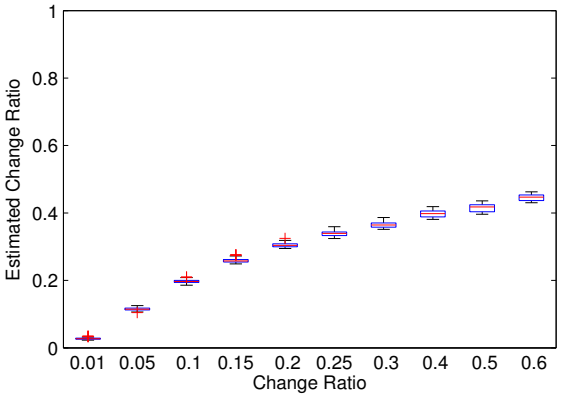
(c) Replacements (of nodes weighting 1-500, with 1-3 real trees with total weight 1-200) (*pq*-grams).



(d) Replacements (of nodes weighting 1-500, with 1-3 real trees with total weight 1-200) (*pql*-grams).

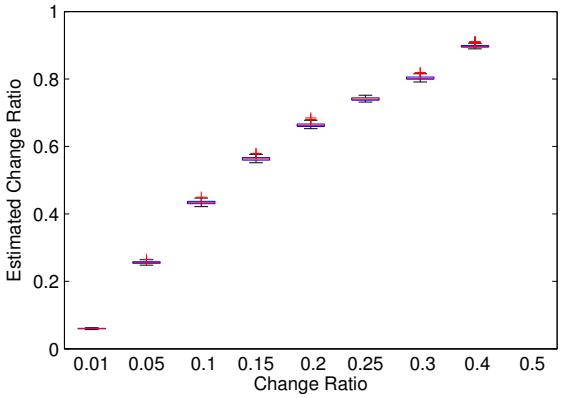


(e) Renames (to a name used by a node at the same nesting depth in the original document) (*pq*-grams).

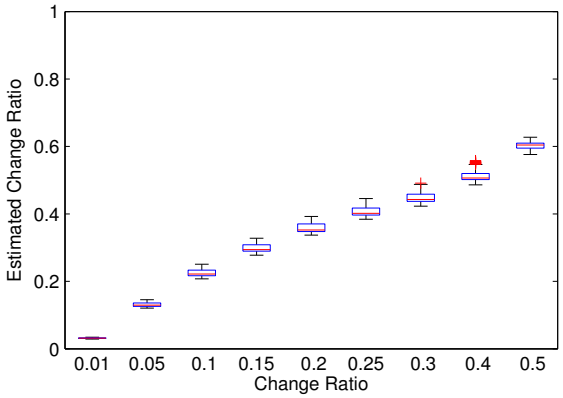


(f) Renames (to a name used by a node at the same nesting depth in the original document) (*pql*-grams).

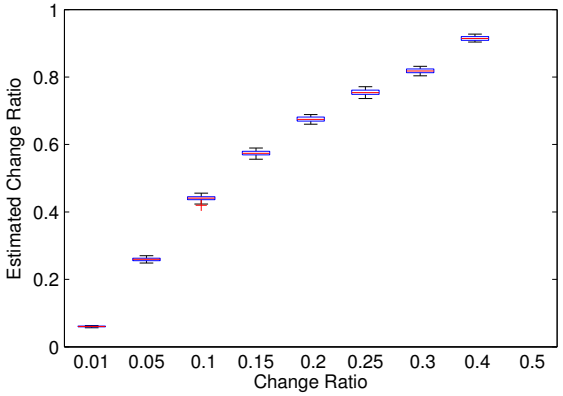
Figure 3.9: Tree PUL edit-distance estimation with *pq*-grams and *pql*-grams (part 2).



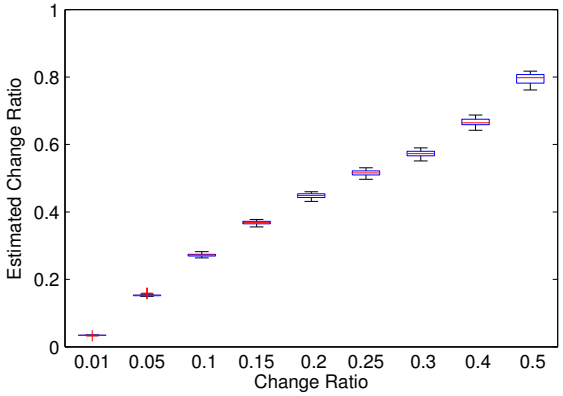
(a) Value replacements (with a value used in the original document) (*pq*-grams).



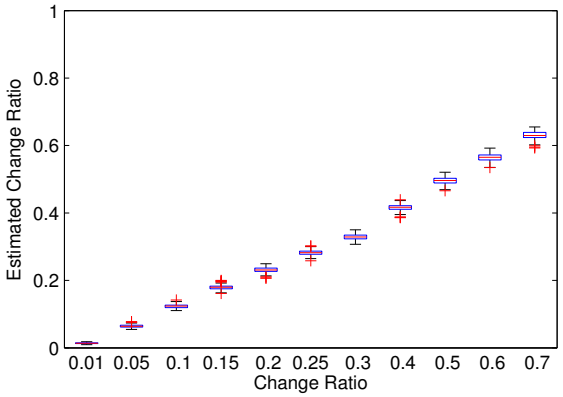
(b) Value replacements (with a value used in the original document) (*pql*-grams).



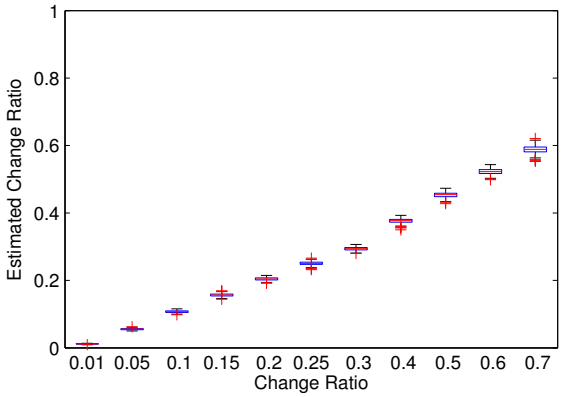
(c) Value replacements (with a random value) (*pq*-grams).



(d) Value replacements (with a random value) (*pql*-grams).

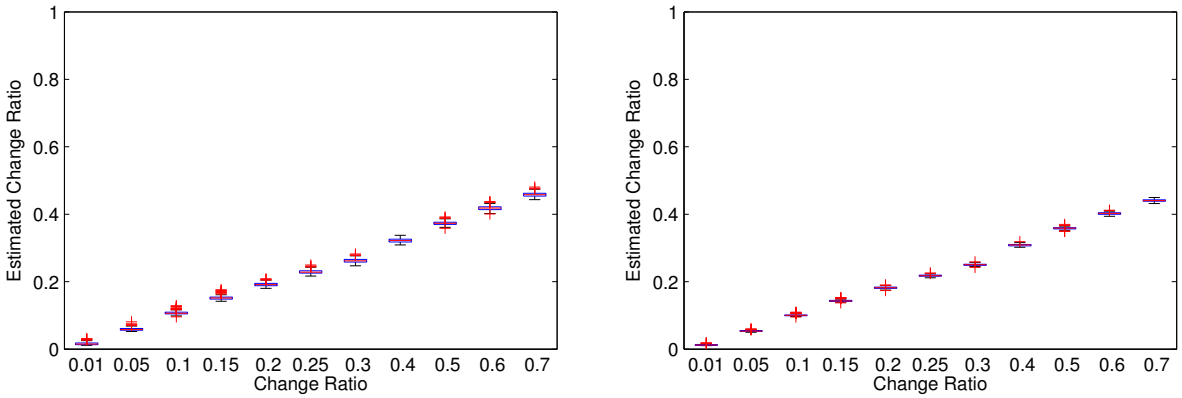


(e) Deletions (removed subtree weight 1-500) (*pq*-grams).



(f) Deletions (removed subtree weight 1-500) (*pql*-grams).

Figure 3.10: Tree PUL edit-distance estimation with *pq*-grams and *pql*-grams (part 3).



(a) Random (*pq*-grams).

(b) Random (*pql*-grams).

Figure 3.11: Tree PUL edit-distance estimation with *pq*-grams and *pql*-grams (part 4).

	***pq*-gram estimation**		***pql*-gram estimation**	
Change Ratio	**Mean**	**Standard dev.**	**Mean**	**Standard dev.**
0.01	0.029572	0.026947	0.016702	0.008945
0.05	0.124398	0.107564	0.074867	0.035163
0.10	0.213398	0.169629	0.135634	0.056661
0.20	0.338368	0.231525	0.231765	0.078262
0.30	0.425672	0.255584	0.305835	0.086130
0.40	0.492553	0.263406	0.365108	0.089523
0.50	0.556594	0.278110	0.418311	0.100024

Table 3.2: PUL change ratio estimation summary, random PULs.

random non-homogeneous PULs. The quality of the chosen matches for entire sequences of children is considered in Section 3.7.3. Here we consider the worst-case scenario: in each sequence all subtrees are very similar to each other (*e.g.*, all DBLP `article` elements or all `auction` XMark elements), and modifications are roughly uniformly distributed among subtrees.

For each generated pair of sequences we contrast the cost difference between the edit-script generated by PUL-Diff ignoring all perfect matches (to simulate matching two sequences of unmatched subtrees), the edit-script generated by PUL-Diff, and an optimal edit-script.

Results for different change ratios, different sequence lengths (from 10 to 500), and different values for k, are reported in Figures 3.12, 3.13, and 3.14. From the figures it emerges that the change ratio does not significantly influence the results. In each plot the left (resp. right) side represents the result obtained without (resp. with) perfect matches.

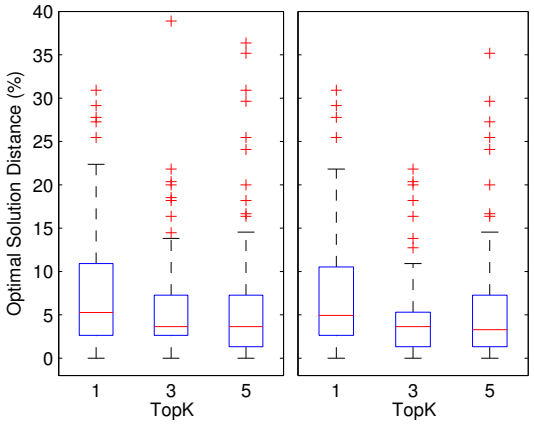
Independently from sequence length and value of k, the selected matches are on average very precise. Indeed, in no cases the median of the cost difference w.r.t. the optimal solution is always below 5%. Increasing the value of k above 3, when perfect matches are employed, has negligible impact. When perfect matches are not employed, increasing the value of k to about half the length of the sequence yields to edit-scripts as expensive as those identified with perfect matches. However, the average distance between the edit-script obtained with/without perfect matches is extremely small (1-2% w.r.t. the optimal edit-script). Surprisingly, increasing the length of the sequence *reduces* the distance between the computed edit-scripts and the optimal one, thanks to the HIPS algorithm.

Results for extremely long sequences (from 1000 to 5000) are instead reported in Figure 3.15. When using perfect matches, the distance (from the optimum) of the obtained edit-scripts still decreases as the sequence length increases and values of k above 3 still have a negligible impact. When perfect matches are not employed, the distance of obtained edit-scripts from the optimum increases with sequences of 1000 subtrees, to decrease again with sequences of 5000 subtrees. We can observe that the match detection based on *pql*-gram is less sensitive with change ratios near 0.3. In any case, even a moderate value of k is still sufficient to keep the average distance well below 5%.

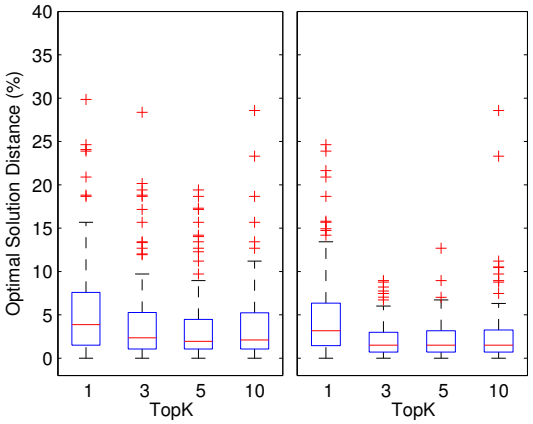
Therefore, the proposed *pql*-gram-based distance metric, when paired with an HIPS-based top k matches pruning algorithm, is very reliable. Moreover, we stress that PUL-Diff benefits from perfect matches, and thus the length of the sequences compared by the algorithm is much shorter.

3.7.3 Performance Test

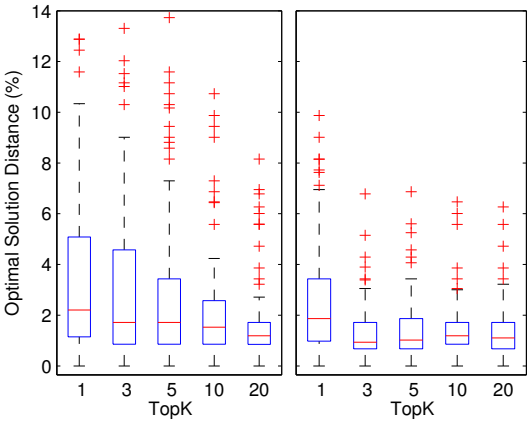
This test aims at empirically determining the time complexity of PUL-Diff for different document sizes and change ratios, as well as contrasting the edit-script cost and computational time of PUL-Diff with the state of the art. The PULs generating the target documents contain randomly



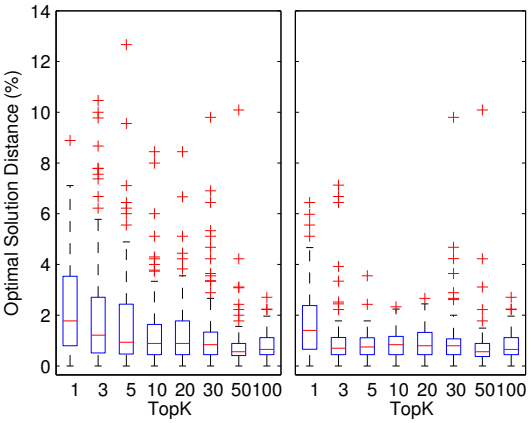
(a) Sequence length 10.



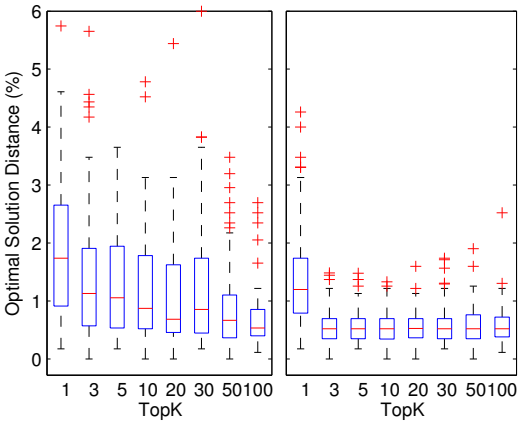
(b) Sequence length 25.



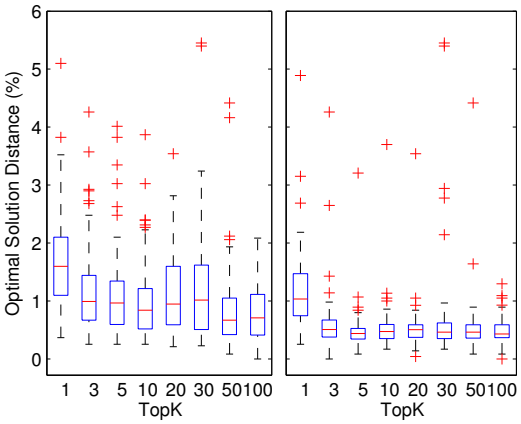
(c) Sequence length 50.



(d) Sequence length 100.

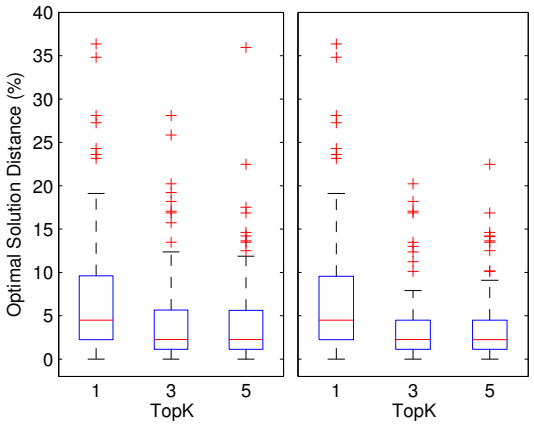


(e) Sequence length 250.

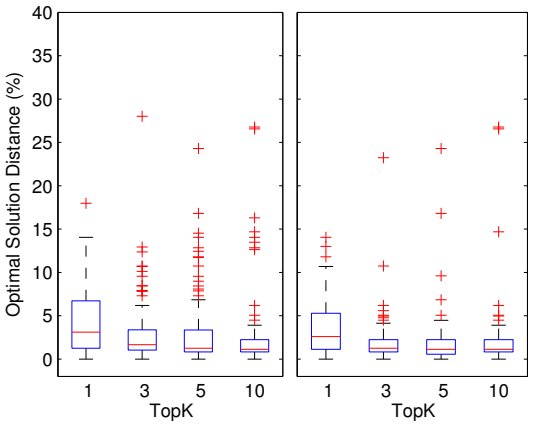


(f) Sequence length 500.

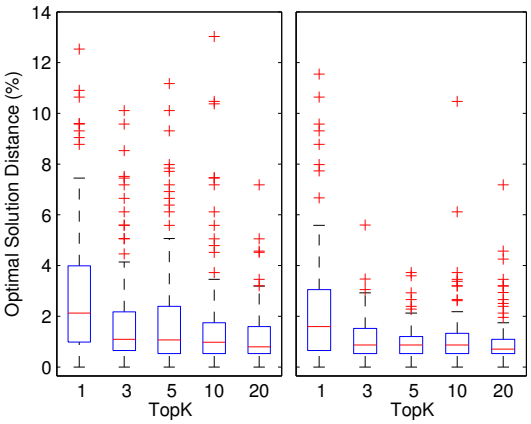
Figure 3.12: Sequence matching with change ratio 0.3, cost distance from an optimal solution.



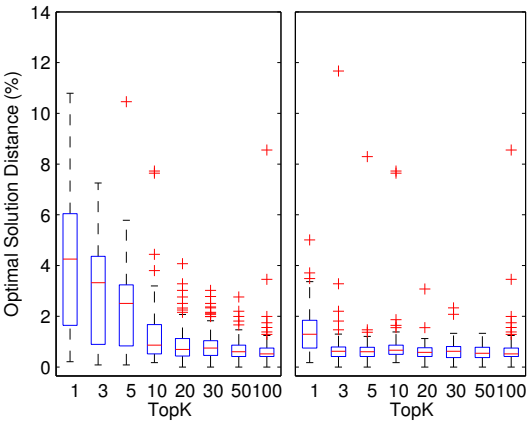
(a) Sequence length 10.



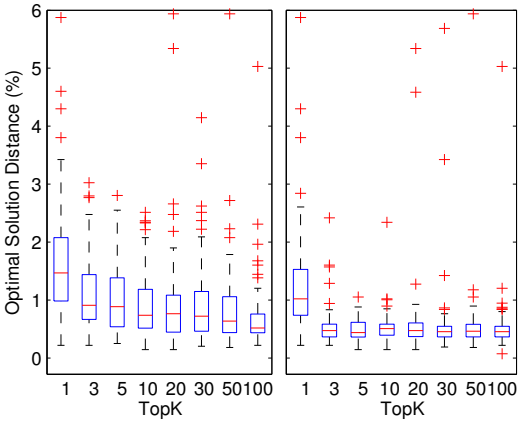
(b) Sequence length 25.



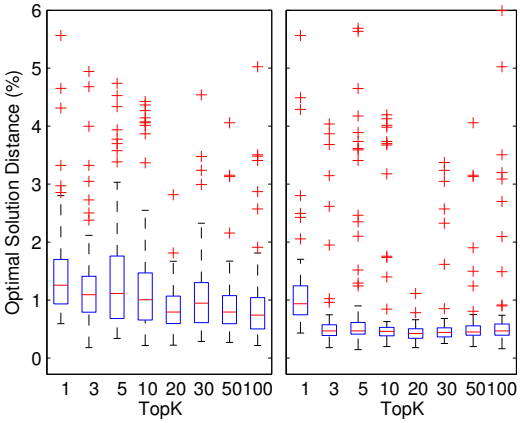
(c) Sequence length 50.



(d) Sequence length 100.

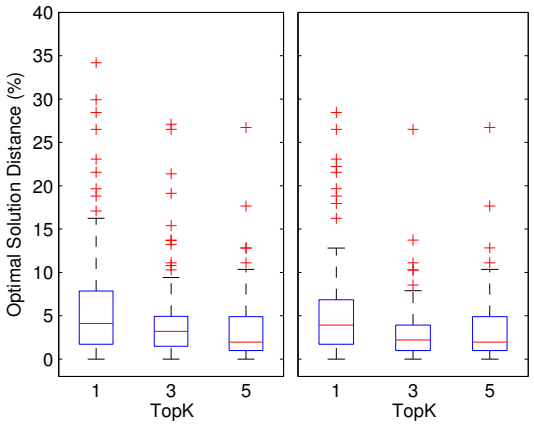


(e) Sequence length 250.

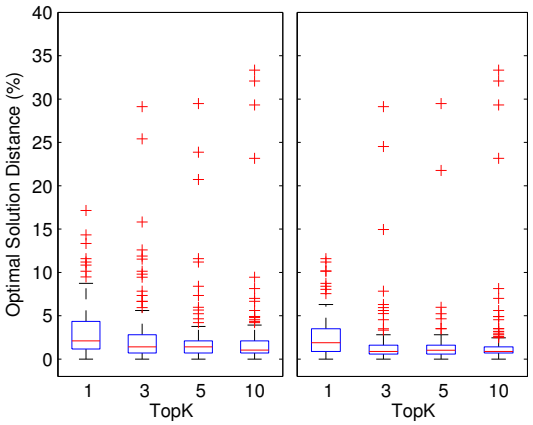


(f) Sequence length 500.

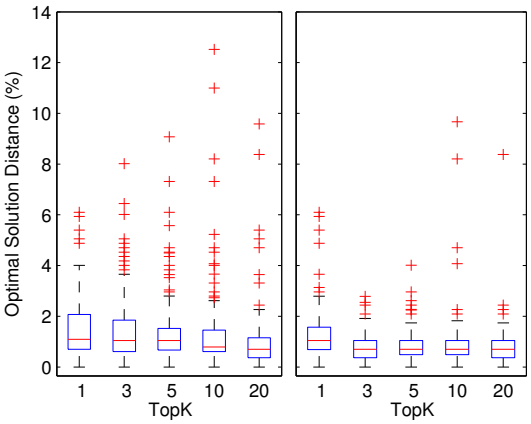
Figure 3.13: Sequence matching with change ratio 0.5, cost distance from an optimal solution.



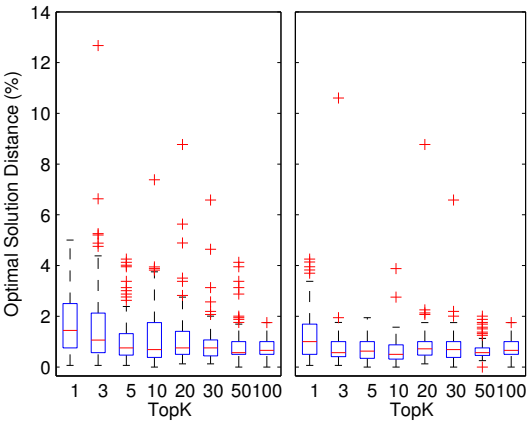
(a) Sequence length 10.



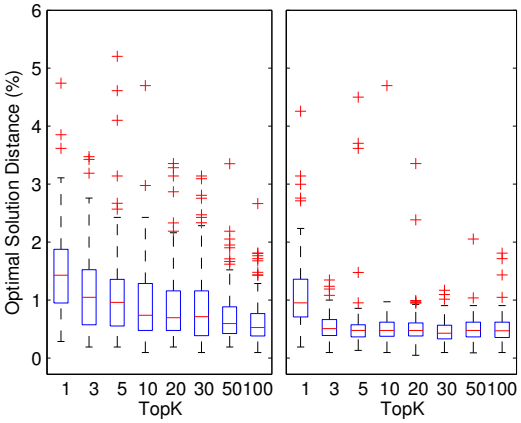
(b) Sequence length 25.



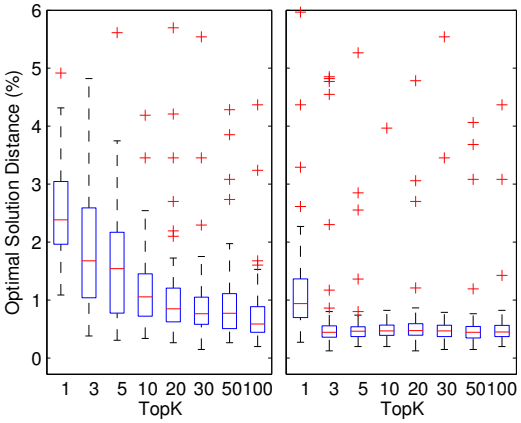
(c) Sequence length 50.



(d) Sequence length 100.

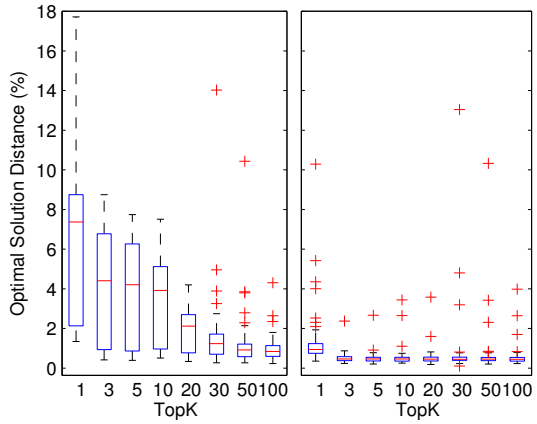


(e) Sequence length 250.

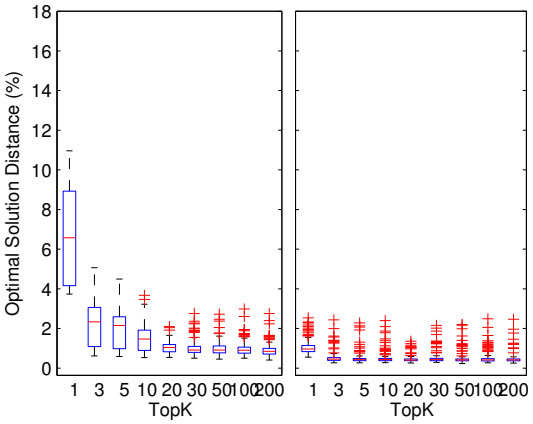


(f) Sequence length 500.

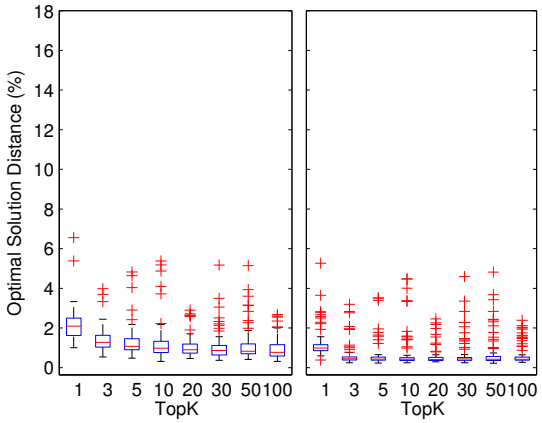
Figure 3.14: Sequence matching with change ratio 0.7, cost distance from an optimal solution.



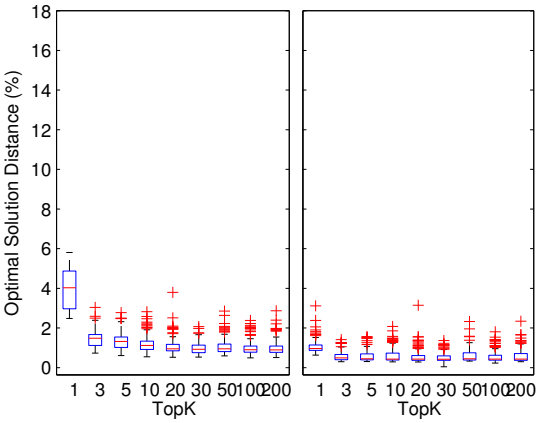
(a) Sequence length 1000, change ratio 0.3.



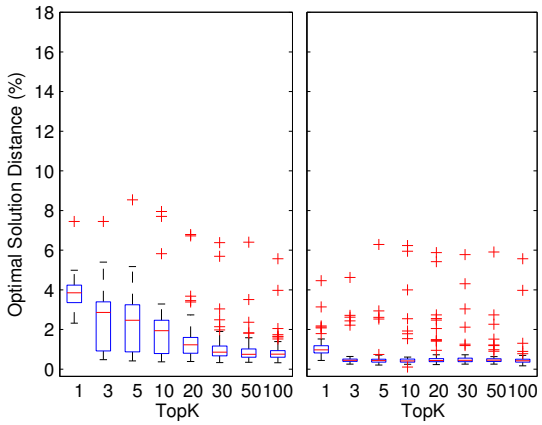
(b) Sequence length 5000, change ratio 0.3.



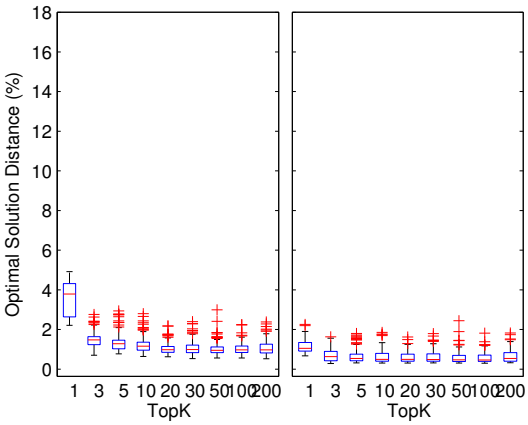
(c) Sequence length 1000, change ratio 0.5.



(d) Sequence length 5000, change ratio 0.5.



(e) Sequence length 1000, change ratio 0.7.



(f) Sequence length 5000, change ratio 0.7.

Figure 3.15: Sequence matching for very long sequences, cost distance from an optimal solution.

generated del, $\texttt{ins}^d$, repN, ren and repV operations.

We identified several interesting XML differencing tools for ordered models in the literature, including XyDiff [CAM02], DeltaXML [Fon01], MMDiff [Cha99], XMDiff [Cha99], and RTED [PA11], the best performing exact algorithm. On our test machine RTED requires more than 3 minutes and 14GB of RAM for differencing two 256KB documents. Other exact algorithms (*i.e.*, MMDiff and XMDiff) require even more time.

For this reason, we restricted our attention to XyDiff (C++ version) and DeltaXML 6.4.1 (Linux 14-days trial version, limited to 1 million nodes, roughly, 7MB). The edit-script produced by these algorithms can be contrasted with ours without unfair advantages. Specifically, w.r.t. our cost model, the relevant differences are that the repC operation exist only in PUL-Diff and that a move operation exists only in XyDiff. Since in this test we disabled the generation of repC operations, and since XyDiff devotes a great effort to minimize the number of moves, we just consider the cost of XyDiff move operations as the cost of the equivalent deletion and insertion operations. Moreover, we double checked that the XyDiff edit-script costs are fair by contrasting the presented results with the results obtained disabling the generation of moves in the PULs used to produce the target documents, and employing only randomly generated names, values, and inserted/replacement subtrees to almost avoid move operations in the XyDiff edit-scripts.

We report in Figure 3.16 the cost difference between a minimum-cost edit-script (the one used to generate the target document) and the cost of the edit-script generated by PUL-Diff, XyDiff, and DeltaXML. We consider different change ratios and document sizes. As can be observed in the figure, independently from the change ratio, both PUL-Diff and DeltaXML produce almost minimal edit-scripts, with a slight advantage for PUL-Diff. XyDiff, instead, produces far worse results. With very small documents (up to 1MB) its quality is comparable with that of the other two algorithms, but usually the generated edit-script is far from optimal. This result does not seem to depends on the XyDiff algorithm implementation as [CAH02, CAM02] consider small documents and report that XyDiff can compute an edit-script 5 times more expensive than the considered reference (on average 2 times more expensive).

In Figure 3.17, we also contrast the time required for differencing two documents with different sizes and change ratios. We can observe that XyDiff is roughly 2 to 5 times slower than PUL-Diff and that both algorithms have an almost linear time complexity. As expectable, the change ratio and the two document sizes influence the expected computational time. For what concerns PUL-Diff, as the change ratio increases, the number of perfectly matched subtrees decreases, thus increasing the time spent in the top-down refinement stage.

Roughly we can observe that both XyDiff and PUL-Diff are three times slower when the change ratio is 0.9 w.r.t. their results with change ratio 0.01. The result also shows that the excellent quality of the DeltaXML edit-scripts is counterbalanced by the computational time required by the algorithm, which is clearly exponential when the change ratio is 0.3 or higher. Indeed, with change ratio 0.01, DeltaXML is only marginally slower than PUL-Diff, whereas with change

ratio 0.1 it is as slow as XyDiff. With higher change ratios, the time required by DeltaXML becomes exponential in the size of the considered documents.

Part I - Conclusions and Future Work

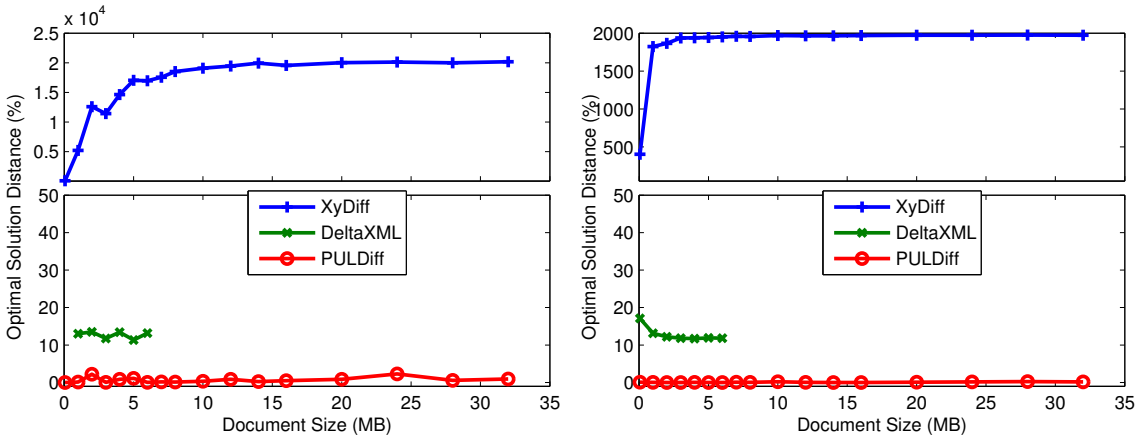
The first part of the thesis focused on change management for XML targeting, specifically, the effects of schema updates on the associated document collection (Chapter 2), and the capability to efficiently detect and synthetise changes among XML documents as PULs (Chapter 3). Both contributions could be subject to improvements and extensions in future works.

For the first topic, node selection constraints for update operations could be refined, for example using *XPath* axes [Jam99] and the other features offered by XQUF. Moreover, support for commutative trees, in which the order of the children of a node is irrelevant, could be added. This feature would allow the formalization of the *all* and *interleave* constructs of XML Schema [Pri04] and Relax NG [CM01], respectively, and the overcome of the need of considering several alternative automata for the INS_{into} operation. Sheaves Automata, introduced in [ZL03], are able to recognize commutative trees and have an expressiveness strictly greater than that of HA considered in our approach. The applicability of these automata in our framework needs to be investigated, given that, as shown in [ZL03], the *inclusion test* is undecidable for Sheaves Automata.

Concerning the second contribution, the choice of PULs as edit-script language influences the algorithm, since no move operator is considered, while internal node relabelings and changes at leaves are detected. No assumptions are made on the trees to be compared (*e.g.*, few duplicate node labels, as in [CRGMW96]). The results of the experimental evaluation against state of the art approaches are very good, both in terms of time and edit-script cost. As future work we plan to extend the proposal for supporting all the node kinds (*e.g.*, attributes and comments), to refine the *pql*-grams based edit-distance estimation with a deeper inquiry of its formal properties, and to perform experiments on the spatial complexity. Another interesting line of research would be to employ, in the window-based matching phase, the technique proposed in [CO14] for finding similar trees, by using *pql*-grams as a tree distance measure.

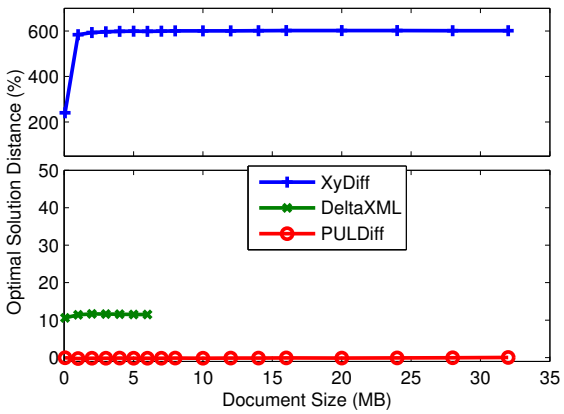
Despite covering different aspects of XML co-evolution, both contributions could be integrated into a single XML co-evolution framework, possibly as an extension of already existing XML evolution tools, such as *EXup* [CGM11b]. The proposed static analysis technique, for instance, could be used as an alternative to the runtime revalidation algorithms employed in *EXup*. PUL-Diff, instead, could be used to efficiently implement an XML document versioning system, that could be paired with existing XML schema versioning systems, enabling in this way a full support for XML co-evolution.

PUL-Diff could be also used to compare manually adapted documents, after a schema update, to

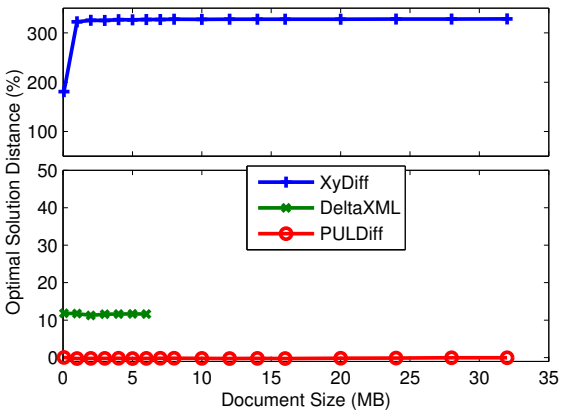


(a) Change ratio 0.01.

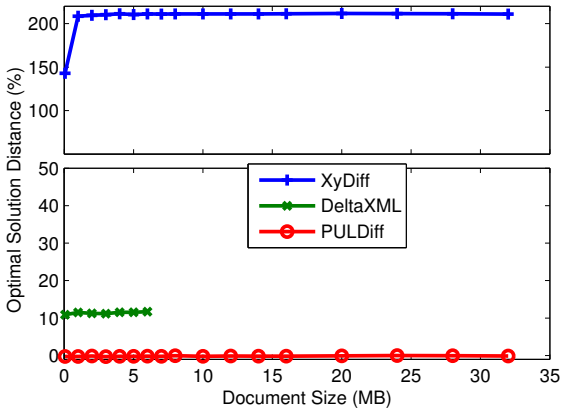
(b) Change ratio 0.1.



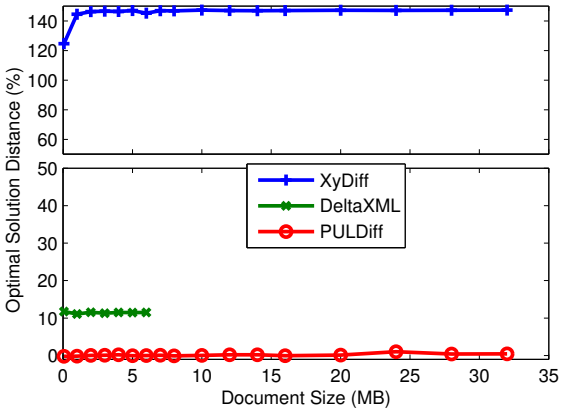
(c) Change ratio 0.3.



(d) Change ratio 0.5.

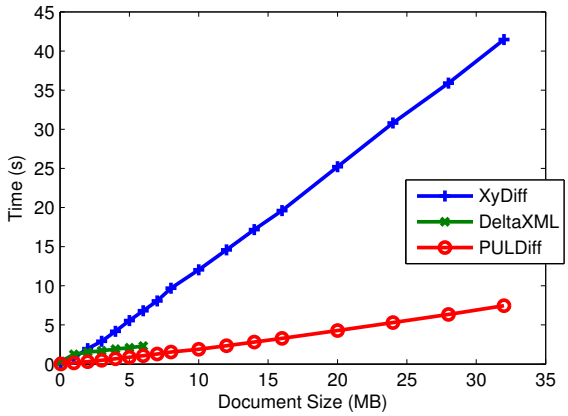


(e) Change ratio 0.7.

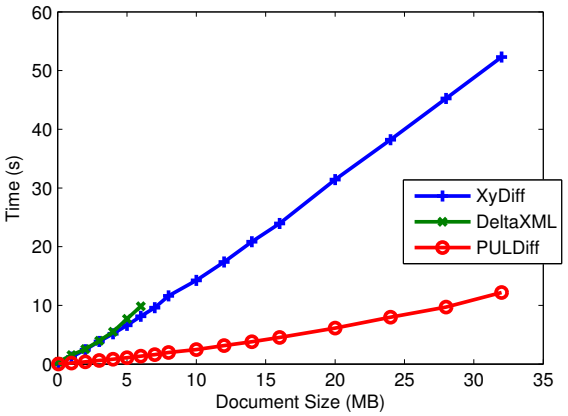


(f) Change ratio 0.9.

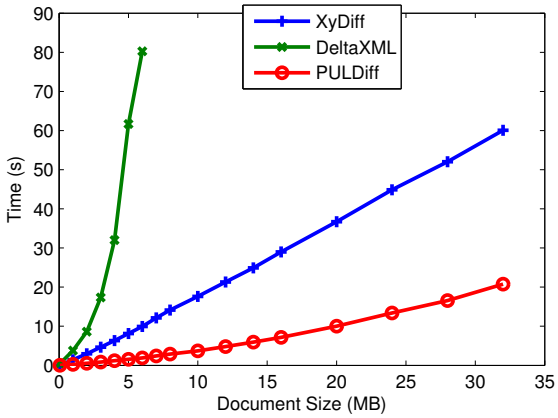
Figure 3.16: Cost distance from an optimal solution.



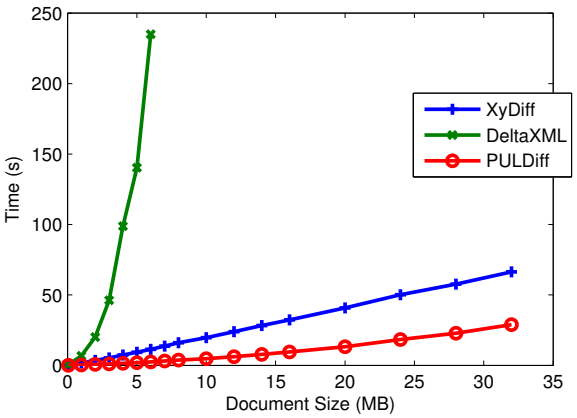
(a) Change ratio 0.01.



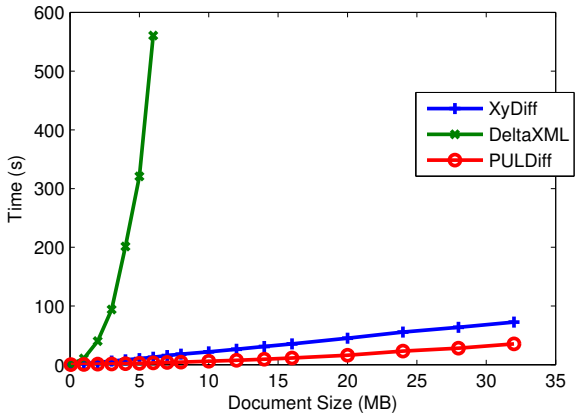
(b) Change ratio 0.1.



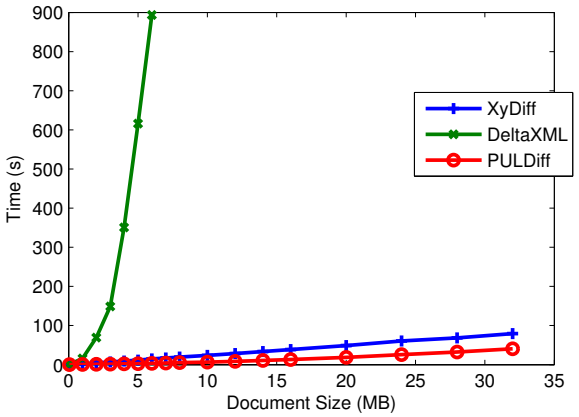
(c) Change ratio 0.3.



(d) Change ratio 0.5.



(e) Change ratio 0.7.



(f) Change ratio 0.9.

Figure 3.17: Differencing time at varying document size.

help the user to extract document adaptations by inspecting the computed edit-script. In addition, a heuristic to automatically extract a sequence of update operations from the computed edit-script(s), generalizing the automatic adaptation approach currently supported in *EXup*, could be devised.

Part II

Ontology Alignment

Chapter 4

Ontology Matching: Basics and Motivations

The aim of this chapter is to provide a general introduction for the second part of the thesis (Section 4.1), and to provide the necessary preliminaries about the main discussed topics, namely Description Logics (Section 4.2), Ontology Matching (Section 4.3), Horn Propositional Logic and Answer Set Programming (Section 4.4), and Graph-Theory (Section 4.5). At the same time, the shared notations and definitions for the remainder of the thesis are also introduced.

Finally, Section 4.6 describes the relevant state of the art, and Section 4.7 provides concrete motivating scenarios for the problem addressed in the second part of the thesis, analyzing practical situations of interest that involve the use of ontology alignments.

4.1 Introduction

In the first part of the thesis we considered techniques relating to XML co-evolution. As already discussed in Section 1.1, change management for XML can be fully addressed considering the syntactical level only, while ontologies and related metadata, due to their underlying formal semantics, force to consider the consequences of any occurring modification, in order to fully understand change effects.

In this setting, the boundaries between development, reasoning and debugging tools are not neat, since all these different aspects are deeply interrelated. Indeed, ontology development tools incorporating versioning and debugging techniques have been proposed (*e.g.*, [JRCHB11b]). This also holds for ontological metadata such as ontology-to-ontology alignments.

For this reason, starting from the principles that semantically define correct ontology-to-ontology

alignments, already identified in literature, we aim at defining detection and debugging techniques, that can be used in many diverse applications, including ontology-to-ontology alignment evolution. The relationships between ontology-to-ontology alignment debugging and adaptation techniques are described in more details in Section 4.7.

Despite debugging techniques are fundamental for ontology evolution, they are subject to more general applications, and have interest on their own. For this reason, after analysing the relationships between the two fields, we will not insist on this aspect any further, and we will uniquely concentrate on the debugging techniques in isolation.

4.2 Description Logics

In this section we cover the basics about Description Logics and Web Ontology Language, starting from [BCM$^+$03, Rud11].

Description Logics (DLs), as already discussed in Section 1.1, originated in the 80's with the aim of proposing ontology languages with a well-defined semantics.

Specifically, DLs is a family of formal knowledge representation languages, that aim at representing the knowledge of an application domain (called the "world"), in an unambiguous way, thus allowing to share the conceptualization of a domain. Firstly, the relevant concepts (also known as classes) for the considered domain are specified, then such domain is finally described by using these concepts to specify properties of objects and individuals.

The DL languages are typically fragments of *First-Order Logic* (FOL), at a varying expressive power and different computational complexity for the associated reasoning tasks, seen as a fundamental service. Despite a sufficient expressive power, DLs present nice computational properties for the standard reasoning services, that are not only guaranteed to be decidable, but are usually tractable in practice.[1]

The different aspects that characterize a specific description logic are the syntactic rules for composing concept and role expressions, as well as the syntactic constraints that prevent the co-existence of particular combinations of axioms.

Each ontology is equipped with a *signature*, that is a vocabulary of legal names for the *entities* appearing in the ontology (namely individuals, concepts and properties). Intuitively, the signature of an ontology plays the role of the alphabet for an automaton (see Chapter 2). Complex concept and role expressions are built on top of atomic entities, of the corresponding kind, belonging to the signature.

[1] Excluding some lightweight DLs, the computational complexity of reasoning services often exceeds polynomial time, but the optimizations and heuristics provided by concrete implementations typically guarantee good performance in practice.

The term *ontology* originates from philosophy, and for our purpose it will be used here as a synonym for *Knowledge Base* [Rud11].

More precisely, the term ontology is employed to refer to a set of axioms, conformant to the syntactic rules and constraints imposed by a fixed DL language $\mathcal{L}$, and built using a signature Σ. An *axiom* is a logical statement relating the entities of the signature through the constructors and connectives allowed by the language $\mathcal{L}$. Intuitively speaking, the axioms are composed by valid concept/role expressions w.r.t. the (formal) grammar characterizing $\mathcal{L}$.

In Definition 4.1 we give the formal definition of the signature of an ontology.

Definition 4.1. The *signature associated with an ontology* $\mathcal{O}$, usually denoted with $Sig(\mathcal{O})$, is the disjoint union of four finite sets: *(i)* N_C, a set of unary symbols called *atomic concepts* or *atomic classes*, *(ii)* N_R, a set of binary symbols called *atomic object properties* or *atomic roles*, *(iii)* N_D, a set of binary symbols called *data properties*, *(iv)* N_I, a set of constant symbols called *individuals*. These sets are denoted with $N_j(\mathcal{O})$, where $j \in \{C, D, R, I\}$. $\triangle$

A signature and an instance of ontology using it is given in Example 4.1.

Example 4.1. Let Σ be a signature composed by the following sets:

- $N_C = \{Vegetarian, RaceHorse, Horse, Unicorn, FlyingAnimal, Animal, Monkey, Baboon, Fruit, Meat\}$
- $N_R = \{eats, isEatenBy\}$
- $N_I = \{charlie, fu, fury, banana, birdy\}$

Let $\mathcal{T}$ a set of axioms defined as follows:

1. $Animal \sqsubseteq Meat$,
2. $RaceHorse \sqsubseteq Horse$,
3. $Horse \sqsubseteq Animal$,
4. $Baboon \sqsubseteq Monkey$,
5. $Monkey \sqsubseteq Animal$,
6. $Unicorn \sqsubseteq FlyingAnimal$,
7. $Unicorn \sqsubseteq Horse$,
8. $Vegetarian \equiv \neg\exists eats.Meat$,

9. $FlyingAnimal \sqcap Horse \sqsubseteq \bot$,
10. $FlyingAnimal \sqsubseteq Animal$,
11. $Bird \sqsubseteq FlyingAnimal$,
12. $Fruit \sqcap Meat \sqsubseteq \bot$,
13. $eats \equiv isEeatenBy^{-}$.

Let also $\mathcal{A}$ be another set of axioms comprising the following elements:

I. $eats(fu, banana)$,

II. $isEatenBy(birdy, fu)$,

III. $Baboon(fu)$,

IV. $RaceHorse(fury)$,

V. $Fruit(banana)$,

VI. $Bird(birdy)$.

An instance of an ontology defined on top of signature Σ is $\mathcal{O} = \langle \Sigma, \mathcal{T} \cup \mathcal{A} \rangle$. $\Diamond$

When the signature of an ontology is omitted, it is implicitly defined as the set of all the symbols appearing in the axioms defining the ontology. The concrete DL language we employ for our definitions is $\mathcal{SROIQ}$,[2] one of the most expressive standard DL languages currently available[3] [HKS06].

$\mathcal{SROIQ}$ is the DL underpinning the *Web Ontology Language* [W3C09] (OWL 2), the widely accepted standard, endorsed by W3C, for specifying DL ontologies in the Semantic Web context. Despite all the experiments of the remainder of the thesis use OWL 2 ontologies and related technologies, for sake of uniformity all the material we present will employ standard nomenclature and notation from the DL community. The tight relationship between OWL 2 and $\mathcal{SROIQ}$ is however well understood [Rud11, KSH12], and all the existing differences between the two relate to features not employed by the techniques we propose.

Before introducing the formal definition of axiom, we need to specify the grammars for valid role and class expressions in $\mathcal{SROIQ}$ (Definition 4.2 and Definition 4.3, respectively).

[2]For a definition of the naming conventions used for DL, the interested reader can refer to [Rud11], Chapter 4.

[3]Note that, for brevity, we do not consider datatypes, functionality constraints and other important DL features, because they are not employed any further in the remainder of the thesis. For this reason any further reference to them is omitted.

Definition 4.2. Given a signature Σ, the following grammar defines *valid role expressions* for $\mathcal{SROIQ}$:

$$\boldsymbol{R} ::= u \mid N_R, \mid N_R^-,$$

where u is the universal role (the Cartesian product of the elements of the domain), and N_R^- is the set of inverse object properties. $\triangle$

Definition 4.3. Given a signature Σ, the following grammar defines *valid class expressions* for $\mathcal{SROIQ}$:

$$\boldsymbol{C} ::= N_C \mid (\boldsymbol{C} \sqcap \boldsymbol{C}) \mid (\boldsymbol{C} \sqcup \boldsymbol{C}) \mid \neg \boldsymbol{C} \mid \bot \mid \top \mid \exists \boldsymbol{R}.\boldsymbol{C} \mid \forall \boldsymbol{R}.\boldsymbol{C} \mid \leq_n \boldsymbol{R}.\boldsymbol{C} \mid \geq_n \boldsymbol{R}.\boldsymbol{C} \mid \exists \boldsymbol{R}.Self \mid \{N_I\},$$

where n is a non-negative integer. $\triangle$

An example of axioms whose right hand side involve complex class and role expressions is given in Example 4.1 by axioms 8 and 13, respectively.

Notice that the *structural restrictions* that any valid $\mathcal{SROIQ}$ ontology must satisfy (that is, constraints between multiple axioms) are omitted because they are not necessary for the understanding of the remainder of the thesis.[4]

The formal definition of axiom is given in Definition 4.4, where we distinguish between *assertional* and *terminological* axioms. Intuitively, assertional axioms (whose set is usually referred to as *ABox*) relate individuals with concepts and roles, while terminological axioms (composing the set of axioms called *TBox* and *RBox*) typically describe the interactions between concepts and roles. An instance of ABox and TBox is provided in Example 4.1.

Definition 4.4. An *axiom* in $\mathcal{SROIQ}$ can be any of the axioms listed in Table 4.2, built using the following grammars for assertional and terminological axioms:

ABox:	$\boldsymbol{C}(N_I)$	$\boldsymbol{R}(N_I, N_I)$	$N_I \approx N_I$	$N_I \not\approx N_I$
TBox:	$\boldsymbol{C} \sqsubseteq \boldsymbol{C}$	$\boldsymbol{C} \equiv \boldsymbol{C}$		
RBox:	$\boldsymbol{R} \sqsubseteq \boldsymbol{R}$	$\boldsymbol{R} \equiv \boldsymbol{R}$	$\boldsymbol{R} \circ \boldsymbol{R} \sqsubseteq \boldsymbol{R}$	$Disjoint(\boldsymbol{R}, \boldsymbol{R})$

$\triangle$

Note that concept/role inclusion is also called (concept/role) *subsumption*.

In the remainder of the thesis we only concentrate on DL ontologies, that from now on are simply called ontologies. Furthermore, we do not differentiate between TBox and RBox, and we will refer to the terminological axioms simply as TBox.

The notion of ontology, given in Definition 4.5, can be defined on top of that of signature and axiom.

[4]The interested reader can find all the details in [KSH12].

Definition 4.5. Given a fixed language $\mathcal{L}$, two finite sets $\mathcal{T}$ and $\mathcal{A}$ of terminological and assertional axioms (respectively) expressed in $\mathcal{L}$, over the elements of a signature Σ, we say that the pair $\mathcal{O} = \langle \Sigma, \mathcal{T} \cup \mathcal{A} \rangle$ is an *ontology* written in $\mathcal{L}$. $\triangle$

One of the main strenghts of DLs is their well-defined associated semantics. DLs' model-theoretic semantics is defined in a compositional manner, that is, the semantics of complex (*i.e.*, not atomic) elements is defined in terms of the semantics of their constituents. In this way, the semantics of an ontology is defined by the semantics of its axioms, as well as they are defined by the semantics of their role and class expressions.

We are now ready to formally define DL semantics in terms of the notion of *interpretation*.

Definition 4.6. An *interpretation* $\mathcal{I}$ is composed by a set $\Delta^{\mathcal{I}}$, called the domain of $\mathcal{I}$, and a interpretation function $\cdot^{\mathcal{I}}$ that maps each atomic concept C into to the set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each atomic role r to the binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual name a to an element of the domain $a^{\mathcal{I}}$. $\triangle$

In Table 4.1 and Table 4.2, borrowed from [KSH12], the semantics associated with the available axioms, connectives and constructors for $\mathcal{SROIQ}$, is detailed.

		Syntax	**Semantics**
Individuals	individual name	a	$a^{\mathcal{I}}$
Roles	atomic role	r	$r^{\mathcal{I}}$
	inverse role	r^-	$\{\langle x, y\rangle \mid \langle y, x\rangle \in r\}$
	universal role	u	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Concepts	atomic concept	A	$A^{\mathcal{I}}$
	intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	complement	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
	top concept	$\top$	$\Delta^{\mathcal{I}}$
	bottom concept	$\bot$	$\emptyset$
	existential restriction	$\exists r.C$	$\{x \mid \exists y \in C^{\mathcal{I}} \wedge \langle x, y\rangle \in r^{\mathcal{I}}\}$
	universal restriction	$\forall r.C$	$\{x \mid \forall \langle x, y\rangle \in r \Rightarrow y \in C^{\mathcal{I}}\}$
	at-least restriction	$\geq_n r.C$	$\lvert\{x \mid \exists y \in C^{\mathcal{I}} \wedge \langle x, y\rangle \in r^{\mathcal{I}}\}\rvert \geq n$
	at-most restriction	$\leq_n r.C$	$\lvert\{x \mid \exists y \in C^{\mathcal{I}} \wedge \langle x, y\rangle \in r^{\mathcal{I}}\}\rvert \leq n$
	local reflexivity	$\exists r.Self$	$\{x \mid \langle x, x\rangle \in r^{\mathcal{I}}\}$
	nominal	a	$\{a^{\mathcal{I}}\}$

Table 4.1: Syntax and semantics of the different connectives and constructors for $\mathcal{SROIQ}$, with $a \in N_I$, $A \in N_C$, C, D are class expressions (see Definition 4.3), r, s role expressions (Definition 4.2).

		Syntax	Semantics
ABox	concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
	role assertion	$r(a,b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$
	individual equality	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
	individual inequality	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
TBox	concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
	concept equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
RBox	role inclusion	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
	role equivalence	$r \equiv s$	$r^{\mathcal{I}} = s^{\mathcal{I}}$
	complex role inclusion	$r_1 \circ r_2 \sqsubseteq s$	$r_1^{\mathcal{I}} \circ r_2^{\mathcal{I}} = s^{\mathcal{I}}$
	role disjointness	$Disjoint(r,s)$	$r^{\mathcal{I}} \cap s^{\mathcal{I}} = \emptyset$

Table 4.2: Syntax and semantics of $\mathcal{SROIQ}$ axioms, with $a, b \in N_I$, C, D are class expressions and r, s are role expressions.

After the definition of the semantics of all the syntactical elements, we are now ready to introduce the notion of axiom *satisfiability* w.r.t. an interpretation (Definition 4.7).

Definition 4.7. We say that an axiom α holds in an interpretation $\mathcal{I}$ (or alternatively that $\mathcal{I}$ *satisfies* α), denoted as $\mathcal{I} \models \alpha$, iff the interpretation function associated with $\mathcal{I}$ does not violate the contraints imposed by α. When an interpretation $\mathcal{I}$ *does not satisfy* an axiom α we write $\mathcal{I} \not\models \alpha$. △

In Definition 4.8 we introduce the notion of *model* of an ontology.

Definition 4.8. Any interpretation $\mathcal{I}$ satisfying all the constraints imposed by the axioms composing an ontology $\mathcal{O}$, written $\mathcal{I} \models \mathcal{O}$, is called a *model* of $\mathcal{O}$. The set of models of $\mathcal{O}$ is denoted as $Mod(\mathcal{O})$. △

Example 4.2 gives an example of interpretation that is a model, and others that are not.

Example 4.2. Consider the ontology $\mathcal{O}$ of Example 4.1. Let $\mathcal{I}$ be an interpretation such that $\Delta^{\mathcal{I}} = \{'fu', 'fury', 'charlie', 'banana', 'birdy'\}$ and $\cdot^{\mathcal{I}}$ is defined as follows:

1. the interpretation of each individual is the constant having the same name (*e.g.*, $fu^{\mathcal{I}} = 'fu'$),
2. $Baboon^{\mathcal{I}} = Monkey^{\mathcal{I}} = \{'fu'\}$,
3. $Horse^{\mathcal{I}} = RaceHorse^{\mathcal{I}} = \{'fury'\}$,
4. $Bird^{\mathcal{I}} = FlyingAnimal^{\mathcal{I}} = \{'birdy'\}$,
5. $Animal^{\mathcal{I}} = Meat^{\mathcal{I}} = \{'fu', 'fury', 'birdy'\}$,

6. $Banana^{\mathcal{I}} = Fruit^{\mathcal{I}} = \{'banana'\}$,

7. $Vegetarian^{\mathcal{I}} = \{'fury'\}$,

8. $eats^{\mathcal{I}} = \{\langle 'fu', 'banana'\rangle, \langle 'fu', 'birdy'\rangle\}$,

9. $isEatenBy^{\mathcal{I}} = \{\langle 'banana', 'fu'\rangle, \langle 'birdy', 'fu'\rangle\}$.

Such interpretation is a model for $\mathcal{O}$, because it can be easily verified that it satisfies all the constraints imposed by its axioms. Consider instead another interpretation $\mathcal{I}'$, coinciding with $\mathcal{I}$, except for line 6, that is replaced with $Banana^{\mathcal{I}} = Fruit^{\mathcal{I}} = \{'banana', 'fury'\}$. $\mathcal{I}'$ is not a model for $\mathcal{O}$, because it violates axiom 12, that states that $Fruit$ and $Meat$ are disjoint concepts. Consider another interpretation $\mathcal{I}''$, again coinciding with $\mathcal{I}$, but including constant $'fu'$ in the interpretation of concept $Vegetarian$. $\mathcal{I}''$ is not a model for $\mathcal{O}$, because from axioms 11, 10, and 1, we know that any $Bird$ is $Meat$, from axiom VI we know that $birdy$ is a $Bird$, and axiom II tells us that fu eats $birdy$. Therefore, given that fu eats meat (*i.e.*, an individual of concept $Meat$), then axiom 8 states that fu cannot be vegetarian (*i.e.*, it cannot belong to concept $Vegetarian$). ◊

Definition 4.9 introduces the notion of axiom *entailment* w.r.t. an ontology.

Definition 4.9. Given an axiom α and an ontology $\mathcal{O}$, we say that $\mathcal{O}$ *entails* α, denoted as $\mathcal{O} \models \alpha$ iff α is satisfiable in any model of $\mathcal{O}$. More formally, $\mathcal{O} \models \alpha$ iff for any model $\mathcal{I}$ in $Mod(\mathcal{O})$ we have that $\mathcal{I} \models \alpha$ holds. If $\mathcal{O}$ *does not entail* α we write $\mathcal{O} \not\models \alpha$. △

Example 4.3. Consider the ontology $\mathcal{O}$ of Example 4.1. It can be easily shown that axiom $\alpha = Baboon \sqsubseteq Animal$ is entailed by $\mathcal{O}$, exploiting the transitivity of subsumpion relation and axioms 4 and 5. ◊

Finally, Definition 4.10 introduces the notion of ontology (un-)satisfiability.

Definition 4.10. Given an ontology $\mathcal{O}$, we say that $\mathcal{O}$ is *satisfiable* iff $Mod(\mathcal{O}) \neq \emptyset$, *unsatisfiable* otherwise. A *(un-)satisfiable ontology* is also called *(in-)consistent*. △

Example 4.4. Consider the ontology $\mathcal{O}$ of Example 4.1. By the interaction of axioms 6, 7 and 9, it is evident that concept $Unicorn$ can only be interpreted as the empty set, nonetheless the ontology is satisfiable. Instead, if we add the assertion $\alpha = Unicorn(charlie)$, then the ontology turns unsatisfiable, because the constraints imposed by axioms 6, 7 and 9, and the new assertion α, are incompatible and cannot be satisfied at the same time, thus preventing the existence of a model for $\mathcal{O}$. ◊

Finally, Definition 4.11 gives the definition of concept and role unsatisfiability, and ontology incoherency.

Definition 4.11. A *concept* C (resp. role r) belonging to the signature of an ontology $\mathcal{O}$, is *unsatisfiable* w.r.t. $\mathcal{O}$ iff for each model $\mathcal{I} \in Mod(\mathcal{O})$ we have that $C^{\mathcal{I}} = \emptyset$ (resp. $r^{\mathcal{I}} = \emptyset$). In this case, $\mathcal{O}$ is said to be an *incoherent ontology*. Furthermore, if an axiom $C(a)$ exists in $\mathcal{O}$ (resp. $r(a, b)$), $\mathcal{O}$ is said to be unsatisfiable. $\triangle$

An example of unsatisfiable entity is given by concept $Unicorn$ in the ontology of Example 4.1.

4.2.1 Reasoning Services for DLs

Reasoning by inference, that is the capability of inferring implicit knowledge from the explicit one, is a fundamental characteristic of DLs. Differently from FOL, standard reasoning services are (required to be) decidable. The tractability of such reasoning services varies from DL to DL, because it highly depends on the expressive power of the specific DL. For this reason the research community, while investigating increasingly more expressive DL languages, also pursued the study of lightweight DLs (the most notable examples are the DL-Lite family [CDL$^+$09] and the $\mathcal{EL}$ family [BBL05, BBL08]).

In what follows we briefly describe the main (standard) reasoning services for DLs, that are:

- ontology satisfiability (Definition 4.12),
- entailment checking (Definition 4.13),
- concept satisfiability (Definition 4.14),
- instance retrieval (Definition 4.15),
- ontology classification (Definition 4.16).

Definition 4.12. *Ontology satisfiability* is a *reasoning task* that aims at checking if the set of models for a given ontology is empty. More formally, ontology satisfiability consists in checking, for a given ontology $\mathcal{O}$, whether $Mod(\mathcal{O}) \neq \emptyset$ holds. $\triangle$

Definition 4.13. *Entailment checking* is a *reasoning task* that aims at testing whether an axiom α is entailed by a set of axioms $\mathcal{O}$. More formally, given α and $\mathcal{O}$ as input, entailment checking tests if for each model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models \alpha$ holds. $\triangle$

Definition 4.14. *Concept satisfiability* is a *reasoning task* that, given an ontology $\mathcal{O}$ and a concept C in its signature, aims at testing if at least a model of $\mathcal{O}$ intreprets C as a nonempty set. More formally, given an ontology $\mathcal{O}$ and a concept C in its signature, concept satisfiability reduces at testing whether $C^{\mathcal{I}} \neq \emptyset$, for at least one model in $Mod(\mathcal{O})$. $\triangle$

Note that concept satisfiability is usually employed also for the corresponding reasoning task that aims at checking satisfiability for roles.

Definition 4.15. *Instance retrieval* is a *reasoning task* that, given an ontology $\mathcal{O}$ and a concept C in its signature, aims at deriving all the named individuals that, for any model of $\mathcal{O}$, belong to concept C. More formally, given an ontology $\mathcal{O}$ and a concept C in its signature, instance retrieval retrieves all the $a \in N_I(\mathcal{O})$ such that, for each model $\mathcal{I}$ of $\mathcal{O}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$. △

An example of the retrieval of the instances of a concept is provided by Example 4.5.

Example 4.5. Consider the ontology $\mathcal{O}$ of Example 4.1. The result of instance retrieval for concept $Animal$ is equivalent to the set $\{fu, fury, birdy\}$. ◊

Definition 4.16. Given an ontology $\mathcal{O}$, *ontology classification* is a *reasoning task* that aims at computing the preorder $\sqsubseteq_{\mathcal{O}}$ associated with the subsumption relation between the elements of $N_C(\mathcal{O})$ (resp. $N_R(\mathcal{O})$). More formally, $C \sqsubseteq_{\mathcal{O}} D$ iff $\mathcal{O} \models C \sqsubseteq D$ (resp. $R \sqsubseteq_{\mathcal{O}} S$ iff $\mathcal{O} \models R \sqsubseteq S$). △

Example 4.6 provides an instance of the classification reasoning task.

Example 4.6. Consider the ontology $\mathcal{O}$ of Example 4.1. The result of the classification of its concepts is given in Figure 4.1. Note that the unsatisfiability of concept $Unicorn$ is reflected by the arc from $Unicorn$ to $\bot$, and that thanks to the application of the transitive reduction on the computed DAG, the arcs from $Unicorn$ to $Horse$ and to $FlyingAnimal$ are removed. ◊

Usually the preorder computed by ontology classification is encoded as a directed acyclic graph for visualization purpose, where the vertices are the elements of $N_C(\mathcal{O})$ (resp. $N_R(\mathcal{O})$), and arcs represent the subsumption relation. In order to reduce the redundant information, an additional step, consisting in the computation of the transitive reduction of the graph, is commonly applied.

4.3 Ontology Matching

In this section we provide a brief survey over the main elements of ontology matching. In Section 4.3.1 we give a formal definition of ontology-to-ontology mapping and alignment (adapted from [ES10]). Section 4.3.2 introduces the semantics associated with ontology alignments, and contextualizes the semantics we employ among the alternative proposals existing in literature.

Section 4.3.3 briefly introduces and contextualizes the debugging principles for ontology-to-ontology alignments, and motivates our interest in the conservativity principle. Section 4.3.4 then precisely defines the semantic consequences imposed by ontology alignments. In Section 4.3.5 and Section 4.3.6 we analyze in more detail two of the introduced principles, namely *consistency* and *conservativity* principles.

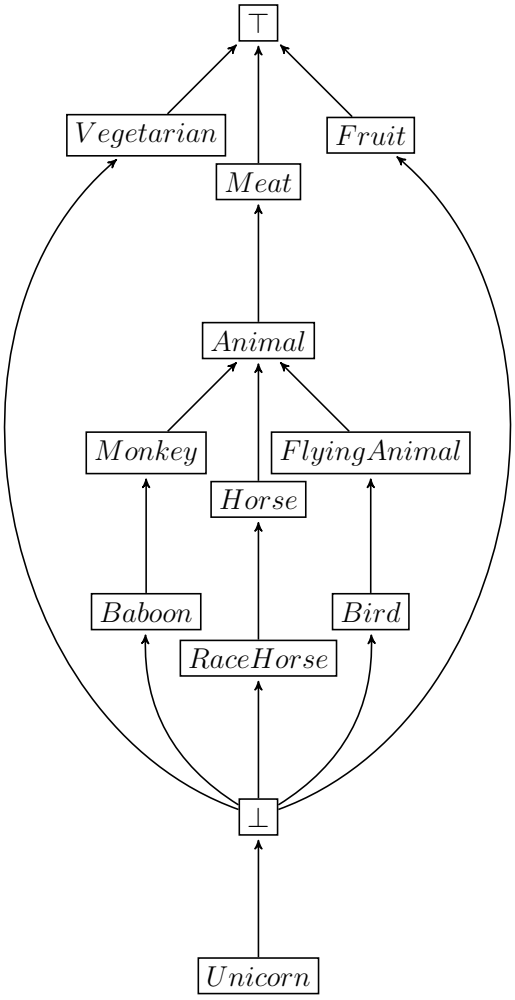


Figure 4.1: Classification output (DAG) for the ontology of Example 4.1.

4.3.1 Ontology-to-ontology Mappings and Alignments

In Definition 4.17 we provide the definition of ontology-to-ontology *mapping* (also called *match* or *correspondence*), while Definition 4.18 introduces the notion of *alignment*.

Definition 4.17. Given two ontologies $\mathcal{O}_1, \mathcal{O}_2$, let Q be a function that defines sets of matchable entities, namely $Q(\mathcal{O}_1) \subseteq Sig(\mathcal{O}_1)$ and $Q(\mathcal{O}_2) \subseteq Sig(\mathcal{O}_2)$. We say that two entities are matched when it is asserted that a correspondence between such two entities exists, w.r.t. the considered semantic relation. That is, no elements in $Sig(\mathcal{O}_i) \setminus Q(\mathcal{O}_i)$ are allowed to be matched, with $i \in \{1, 2\}$. More formally, a *mapping* between entities of two ontologies, namely $\mathcal{O}_1, \mathcal{O}_2$, is a 4-tuple $\langle e, e', r, c\rangle$ such that $e \in Q(\mathcal{O}_1)$ and $e' \in Q(\mathcal{O}_2)$, $r \in \{\sqsubseteq, \sqsupseteq, \equiv\}$ is a semantic relation, and c is a confidence value from a suitable structure $\langle D, \leq\rangle$. △

Usually, the real number unit interval $(0 \dots 1]$ is employed for representing confidence values. Mapping confidence intuitively corresponds to the confidence that the mapping holds, where the confidence increases towards value 1.

The authors of [ES10] also include an identifier for the mapping, that is not needed in our scenario, and it is therefore not considered in our formalization. For the same reason we assume an implicit definition of the function Q, and the validity w.r.t. to it of any proposed mapping. Note that we exclude disjointness from the available semantic relations given that most of the available ontology matchers does not compute this relation.[5] Finally, despite some systems compute equivalence between constant symbols, we do not consider this semantic relation because *instance matching* is out of the scope of the thesis.

In the remainder of the thesis, the term mapping will refer to ontology-to-ontology mapping, and any other ontology mapping kind will be explicitated by need.

Definition 4.18. An *alignment* $\mathcal{M}$ between two ontologies, namely $\mathcal{O}_1, \mathcal{O}_2$, is a set of mappings between $\mathcal{O}_1$ and $\mathcal{O}_2$. △

When no confusion arises the two ontologies are omitted and we simply refer to $\mathcal{M}$. Example 4.7 gives an example of an aligment.

Example 4.7. Let $\mathcal{O}_1$ be the ontology defined in Example 4.1. Let $\mathcal{O}_2$ be the ontology composed by the following axioms: $\{Fish \sqsubseteq Animal, Fish \sqsubseteq \neg Horse, Chimp \sqsubseteq Monkey, SeaHorse \sqsubseteq Fish, DairyProduct \sqsubseteq Food, Egg \sqsubseteq Food, Honey \sqsubseteq Food, RedMeat \sqsubseteq Meat, RedMeat \sqsubseteq Food, Vegan \sqsubseteq \neg\exists eats.Meat \sqcup DairyProduct \sqcup Egg \sqcup Honey\}$.

A possible alignment $\mathcal{M}$ between them could be composed by the following mappings:
$\{\langle Horse_1, Horse_2, \equiv, 1\rangle, \langle Horse_1, SeaHorse_2, \sqsupseteq, 0.6\rangle, \langle Monkey_1, Chimp_2, \equiv, 0.4\rangle,$

[5] The motivation is that, usually, negative constraints are much harder to identify and assess than positive ones [FR12], even in an approximated scenario as ontology matching.

$\langle Monkey_1, Monkey_2, \equiv, 0.9\rangle, \langle Meat_1, Meat_2, \equiv, 0.9\rangle, \langle Meat_1, Food_2, \sqsubseteq, 0.8\rangle,$
$\langle eats_1, eats_2, \equiv, 0.9\rangle\}$. $\Diamond$

4.3.2 Semantics of Mappings and Alignments

The main format to represent mappings has been proposed in the context of the *Alignment API*, and it is called RDF Alignment [DEST11]. This format is the standard for the well-known OAEI campaign. In addition, mappings are also represented as standard subclass, equivalence, and disjointness DL axioms. When mappings are expressed through OWL 2 axioms, confidence values are represented as OWL 2 axiom annotations [JRCHB09].

The representation through standard OWL 2 axioms enables the reuse of the extensive range of OWL 2 reasoning infrastructure that is currently available. Definition 4.19 defines the notion of *aligned ontology*, resulting from the integration of two input ontologies, through an alignment between them.

Definition 4.19. Let $\mathcal{O}_1$, $\mathcal{O}_2$ be two (input) ontologies, and let $\mathcal{M}$ be an alignment between them. The ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} = \langle Sig(\mathcal{O}_1) \cup Sig(\mathcal{O}_2), Axioms(\mathcal{O}_1) \cup Axioms(\mathcal{O}_2) \cup \mathcal{M}\rangle$ is called the *aligned ontology* w.r.t. $\mathcal{O}_1$, $\mathcal{O}_2$, and $\mathcal{M}$. $\triangle$

$\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ is simply called the aligned ontology when no confusion arises. Assuming to that the signature of the aligned ontology is always the union of the signatures of the input ontologies, we often abbreviate the definition of $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ as $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$. In addition, the abbreviated notation $\mathcal{O}^{\mathcal{M}}$ can be employed when the input ontologies are clear from the context. Given that each mapping is translated into a DL axiom, the aligned ontology is again a DL ontology, and therefore the semantics of a mapping (as well as that of alignment and aligned ontology), totally corresponds to the classic semantics of its DL representation.

Note that alternative formal semantics for ontology mappings have been proposed in the literature, such as the one proposed by Euzenat in [Euz07], also known as *reductionistic semantics* [Mei11], and the semantics associated to the so-called *bridge rules* in the context of distributed description logics (DDLs) [BS03, MST09].

For the former, the key ingredient is the presence of an equilising function, relating elements of the local domains of a set of ontologies, to a unique global domain. For convenience of the reader, the main definitions of [Euz07] are reported in what follows (but expressed using our notations). Definition 4.20 formalizes the equilising function.

Definition 4.20. Given a family of interpretations $\langle \mathcal{I}_\mathcal{O}, \Delta^{\mathcal{I}}_{\mathcal{O}}\rangle_{\mathcal{O}\in\Omega}$ of a set of ontologies Ω, an *equilising function* for $\langle \mathcal{I}_\mathcal{O}, \Delta^{\mathcal{I}}_{\mathcal{O}}\rangle_{\mathcal{O}\in\Omega}$ is a family of functions $\gamma = (\gamma_\mathcal{O} \colon \Delta^{\mathcal{I}}_{\mathcal{O}} \to U)_{\mathcal{O}\in\Omega}$ from the domains of interpretation to a global domain of interpretation U. $\triangle$

Reductionistic semantics does not require the relations used in mappings to necessarily belong to the ontology languages, and therefore provides their semantics as in Definition 4.21.

Definition 4.21. Given r an alignment relation and U a global domain of interpretation, r is *interpreted* as a binary relation over U, that is, $r^U \subseteq U \times U$. $\triangle$

Mapping satisfaction is then naturally formulated, in Definition 4.22, through the use of the equilising function γ and the interpretation of the considered semantic relation. Symbol $\circ$ denotes the usual composition of functions.

Definition 4.22. A *mapping* $c = \langle e, e', r, \_ \rangle$ is *satified* for an equilising function γ by two models $\mathcal{I}, \mathcal{I}'$ of $\mathcal{O}, \mathcal{O}'$ iff $\gamma_{\mathcal{O}} \circ \mathcal{I} \in Mod(\mathcal{O})$, $\gamma'_{\mathcal{O}} \circ \mathcal{I}' \in Mod(\mathcal{O}')$, and $\langle \gamma_{\mathcal{O}}(e^{\mathcal{I}}), \gamma_{\mathcal{O}'}(e'^{\mathcal{I}'}) \rangle \in r^U$. $\triangle$

Our context is a special case of the general formalization provided by Euzenat: given two interpretations $\mathcal{I}_1, \mathcal{I}_2$, respectively associated with ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, and having $\Delta_1^{\mathcal{I}}, \Delta_2^{\mathcal{I}}$ as their respective domains, U is defined as $\Delta_1^{\mathcal{I}} \cup \Delta_2^{\mathcal{I}}$, and γ is the identity function. For what concerns the different relations r^U, the standard DL semantics applies, given that all the allowed relations are standard axioms (see Definition 4.17). The semantics we employ is usually called *natural semantics* [Mei11].

For semantics of alignments in the context of DDLs, a relation between each pair of local domains exists. The aim of DDLs is to provide reasoning services between multiple ontologies, connected by means of directional semantic alignments. One of the key features of DDLs is that relations used in correspondences are not axioms in the DL languages, and therefore no entailments can be derived by the interaction between bridge rules and local axioms. However, in Chapter 7 of [BS03], the authors provide a translation of DDL to DL, by means of a complex translation function, that enables to reason in the aligned ontology in a classic way. Therefore, the interaction of local axioms and a translated version of the bridge rules is thus enabled. Thanks to this translation, the techniques proposed in the remainder may also be applicable to DDLs.

4.3.3 Principles for Ontology Alignments

As already discussed in Section 1.3, Jiménez-Ruiz *et al.* [JRCHB11a] identify three principles that ontology mappings need to satisfy: *(i) conservativity principle* (no new semantic consequences involving entities of the same input ontology should be entailed in the aligned ontology that were not entailed by the input ontologies); *(ii) consistency principle* (the aligned ontology needs to be satisfiable); *(iii) locality principle* (the mappings should link entities that have similar *neighbourhoods* in the concept hierarchy).

Violations to these principles (from now on simply violations) may hinder the usefulness of ontology mappings (concrete scenarios and examples are provided in Section 4.7).

Locality principle could be mainly conceived as a mapping quality metric. This principle, however, has also been used in [WX12] for detecting erroneous mappings, exploiting the fact that low-quality mappings are often incorrect.

Consistency principle is the most widely investigated in literature, where tools for detecting and automatically repair mappings leading to logical inconsistency in the aligned ontology (*e.g.*, [JRC11, Mei11]) have been proposed. Violations of the consistency principle are (typically) easy to detect with standard reasoning services, because they always result in incoherent and/or inconsistent (aligned) ontology. However, repairing such violations through standard reasoning services leads to intractability.

Other problematic patterns, not related to the consistency principle, have been already identified in literature [JMSK09, WX12], but no algorithms or details about them are provided. Moreover, the main focus is on detection only, disregarding the repair phase.

A more general formulation of the conservativity principle is known in literature under the name of *conservative extension*, and has been proved that it is already an *EXPTIME*-complete problem for lightweight DL $\mathcal{EL}$ [LW10] and that it is undecidable for DLs as expressive as $\mathcal{ALCQIO}$ [LWW07]. The hardness of the considered problem calls for an investigation of approximate detection and repair techniques, that are proposed in this second part of the thesis.

Specifically, we concentrate on conservativity principle, that aims at capturing the differences in the ontology classification, between the input ontologies and the aligned ontology, an analysis already used in the context of ontology integration [JRCHB09]. Furthermore conservativity principle, despite considering only ontology classification and not the unrestricted problem addressed by conservative extension, is of high interest because classification is one of the most used features in semantic-enabled applications, even when expressive DLs are employed [GPS12, LL13].

Even in our restricted setting, computing a repair by using standard DL reasoning services leads to intractrability. To understand why, we briefly sketch a possible algorithm based on a "what-if" analysis, on the mapping removal. Given a set S of conservativity principle violations, we need to find a subset of the alignment whose removal prevents the entailment, from aligned ontology, of any element of S. The disadvantage of this approach is twofold. On one hand the exponential cardinality of the powerset of the alignment, on the other, the computational complexity of entailment check. For this reason, we again rely on multiple approximated techniques for computing a (possibly suboptimal) repair, based on a combination of graph-theory and Answer Set Programming (Chapter 5) and projections to the Horn Propositional fragment, a less expressive logic equipped with low-complexity reasoning services (Chapter 6). The combination of these different strategies in a multi-strategy approach is investigated in Chapter 7.

Another motivation for studying the problem, is the high frequency of conservativity principle violations, also in high-quality and manually curated alignments such as *UMLS-Metathesaurus*[6]

[6] `http://www.nlm.nih.gov/research/umls/knowledge_sources/metathesaurus/`

[Bod04] (*UMLS*), the most comprehensive effort for integrating biomedical knowledge bases (thesauri and ontologies), as experimentally verified in Sections 5.5.8 and 6.5.2. The occurrence of these problems is frequent also in the alignments generated by top-level matchers, as empirically verified in Sections 5.5.3 and 6.5.3.

The need for algorithms that effectively and efficiently deal with conservativity principle clearly emerges, given the crucial role played by ontologies and ontology alignments, the importance of the conservativity principle for their effectiveness, the intractability of detecting and solving its violations using DL reasoning, the possibly massive size of the input ontologies, and the lack of proposals coping with this problem.

4.3.4 Semantic Consequences of the Integration

The ontology resulting from the integration of two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ via an alignment $\mathcal{M}$ may entail axioms that do not follow from $\mathcal{O}_1$, $\mathcal{O}_2$, or $\mathcal{M}$ alone. These new semantic consequences can be captured by the notion of *deductive difference* [KWW08, KWZ08].

Intuitively, the deductive difference between $\mathcal{O}$ and $\mathcal{O}'$, w.r.t. a signature Σ, is the set of entailments constructed over Σ that do not hold in $\mathcal{O}$, but do hold in $\mathcal{O}'$. The notion of deductive difference, however, has several drawbacks in practice. First, there is no available algorithm for computing it for DLs more expressive than $\mathcal{EL}$, for which it has been proved that the problem is already *EXPTIME*-complete [LW10]. In addition, the problem is undecidable for DLs as expressive as $\mathcal{ALCQIO}$ [LWW07]. Second, the number of entailments in the difference can be huge (even infinite), and so likely to overwhelm users.

In order to avoid the drawbacks of the deductive difference, we rely on the *approximation of the deductive difference*, given in Definition 4.23.

Definition 4.23. Let A, B be atomic concepts (including $\top, \bot$), Σ be a signature, $\mathcal{O}$ and $\mathcal{O}'$ be two ontologies. We define the *approximation* of the Σ-*deductive difference* between $\mathcal{O}$ and $\mathcal{O}'$ (denoted $\mathsf{diff}^{\approx}_{\Sigma}(\mathcal{O}, \mathcal{O}')$) as the set of axioms of the form $A \sqsubseteq B$ satisfying: *(i)* $A, B \in \Sigma$, *(ii)* $\mathcal{O} \not\models A \sqsubseteq B$, and *(iii)* $\mathcal{O}' \models A \sqsubseteq B$. $\triangle$

The proposed approximation only requires to compare the classification hierarchies of ontologies $\mathcal{O}$ and $\mathcal{O}'$ (that can be provided by an OWL 2 reasoner), and it has been successfully used in the past in the context of ontology integration to help users understanding the semantic consequences of this operation [JRCHB09].

Example 4.8 gives an example of axioms belonging to the deductive difference and its approximation.

Example 4.8. Consider the aligned ontology of Example 4.7. An instance of axiom belonging to the deductive difference between $\mathcal{O}_2$ and $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ (w.r.t. the signature $\Sigma = Sig(\mathcal{O}_2)$,

and considering a DL language allowing at least the employed constructors) that do not belong to $\mathsf{diff}^{\approx}_{\Sigma}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ is $\exists eats_2.Meat_2 \sqsubseteq \exists eats.Food_2$. The aforementioned axiom is entailed thanks to the following axiom $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models Meat_2 \sqsubseteq Food_2$, but it cannot belong to the deductive approximation, because it involves complex (anonymous) class expressions. Axiom $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models Meat_2 \sqsubseteq Food_2$, instead, is an element of $\mathsf{diff}^{\approx}_{\Sigma}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$. ◊

4.3.5 Mapping Coherence and Mapping Repair

The consistency principle requires the vocabulary in $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$ to be satisfiable, assuming the union of input ontologies $\mathcal{O}^{\emptyset}_{\mathcal{O}_1,\mathcal{O}_2} = \mathcal{O}_1 \cup \mathcal{O}_2$ does not contain unsatisfiable concepts. Thus $\mathsf{diff}^{\approx}_{\Sigma}(\mathcal{O}^{\emptyset}_{\mathcal{O}_1,\mathcal{O}_2}, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ should not contain any axiom of the form $A \sqsubseteq \bot$, for any $A \in \Sigma = Sig(\mathcal{O}_1) \cup Sig(\mathcal{O}_2)$. In Definition 4.24 we formally define mapping incoherence.

Definition 4.24. A set of *mappings* $\mathcal{M}$ is *incoherent* w.r.t. $\mathcal{O}_1$ and $\mathcal{O}_2$, if there exists a class $A \in Sig(\mathcal{O}^{\emptyset}_{\mathcal{O}_1,\mathcal{O}_2})$, such that $\mathcal{O}^{\emptyset}_{\mathcal{O}_1,\mathcal{O}_2} \not\models A \sqsubseteq \bot$ and $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models A \sqsubseteq \bot$. △

Given that the detection and debugging techniques can be easily extended to deal with properties, they are not explicitly considered here.[7] An incoherent set of mappings $\mathcal{M}$ can be fixed by removing mappings from $\mathcal{M}$. This process is referred to as *mapping repair* (or repair, for short), and is formally introduced in Definition 4.25.

Definition 4.25. Let $\mathcal{M}$ be an incoherent set of mappings w.r.t. $\mathcal{O}_1$ and $\mathcal{O}_2$. A set of mappings $\mathcal{R} \subseteq \mathcal{M}$ is a *mapping repair* for $\mathcal{M}$ w.r.t. $\mathcal{O}_1$ and $\mathcal{O}_2$ iff $\mathcal{M} \setminus \mathcal{R}$ is coherent w.r.t. $\mathcal{O}_1$ and $\mathcal{O}_2$, that is, $\mathcal{O}^{\mathcal{M}\setminus\mathcal{R}}_{\mathcal{O}_1,\mathcal{O}_2} \not\models A \sqsubseteq \bot$, for any $A \in Sig(\mathcal{O}_1) \cup Sig(\mathcal{O}_2)$. △

An incoherent set of mappings can be repaired in many different ways. A trivial repair is $\mathcal{R} = \mathcal{M}$, since an empty set of mappings is trivially coherent (according to Definition 4.24). Nevertheless, the objective is to remove as few mappings as possible. Minimal (mapping) repairs are typically referred to in the literature as *mapping diagnoses* [Mei11] — a term coined by Reiter [Rei87] and introduced to the field of ontology debugging in [SC03]. A repair or diagnosis for an incoherent set of mappings can be computed by extracting the justifications for the unsatisfiable concepts (*e.g.*, [Sch05, KPHS07, SQJH08, HPS08]), and selecting a hitting set of mappings to be removed, following a minimality criteria (*e.g.*, the number of removed mappings, or their combined confidence). However, justification-based technologies do not scale when the number of unsatisfiabilities is large (a typical scenario in mapping repair problems [JRCH12]). To address this scalability issue, mapping repair systems usually compute an *approximate repair* using incomplete reasoning techniques (*e.g.*, [JRC11, Mei11, SFPC13]). An approximate repair

[7] For any property P, a fresh class is introduced, which is posed equivalent to the class expression $\exists P.\top$, and all the mappings between properties are translated into mappings between the fresh classes, as described in [Mei11], Section 5.2.

$\mathcal{R}^{\approx}$ does not guarantee that $\mathcal{M} \setminus \mathcal{R}^{\approx}$ is coherent, but it will (in general) significantly reduce the number of unsatisfiabilities caused by the original set of mappings $\mathcal{M}$.

An example of alignment incoherence and different repair strategies is provided by Example 4.9.

Example 4.9. Consider the aligned ontology of Example 4.7. Given that the concept $SeaHorse$ would be unsatisfiable in $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ (due to the subsumption of $SeaHorse$ with two disjoint classes, namely $Horse$ and $Fish$), alignment $\mathcal{M}$ is incoherent. Depending on the employed repair strategy, one could, for instance:

1. completely remove mapping $\langle Horse_1, Horse_2, \equiv, 1\rangle$,
2. replace $\langle Horse_1, Horse_2, \equiv, 1\rangle$ with its weakening (*i.e.*, $\langle Horse_1, Horse_2, \sqsupseteq, 1\rangle$),
3. remove mapping $\langle Horse_1, SeaHorse_2, \sqsupseteq, 0.6\rangle$,
4. remove axiom $Fish \sqsubseteq \neg Horse$ from $\mathcal{O}_2$,
5. remove axiom $SeaHorse \sqsubseteq Fish$ from $\mathcal{O}_2$.

If mapping confidence is to be minimized, the third strategy would be preferrable among the first three. The last two strategies are not always applicable. Consider, for example, a mapping repair strategy integrated into an ontology matching system. For such systems, a usual assumption is that the input ontologies have to be conceived as immutable, and therefore cannot be altered. $\Diamond$

4.3.6 Conservativity Principle

The conservativity principle, as formulated in [JRCHB11a], states that the integration of the alignment in the two input ontologies (separetely), should not induce any change in their concept hierarchies.

Instead, we consider a generalization of the conservativity principle, requiring that the integrated ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ should not induce any change in the concept hierarchies of the input ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$. That is, the sets $\mathsf{diff}^{\approx}_{\Sigma_1}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ and $\mathsf{diff}^{\approx}_{\Sigma_2}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ must be empty for signatures $\Sigma_1 = Sig(\mathcal{O}_1)$ and $\Sigma_2 = Sig(\mathcal{O}_2)$, respectively.

We further differentiate between violations of the conservativity principle, and we propose a *basic* variant of the conservativity principle, where $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ is required not to introduce new subsumption relationships between concepts from one of the input ontologies, unless they were already involved in a subsumption relationship or they shared a common descendant. In addition to these *basic violations*, we also define violations between concepts already involved in a subsumption relationship (*i.e.*, resulting in an equivalence between them), denoted as *equivalence conservativity principle violations*, or simply *equivalence violations*. The two variants are formally introduced in Definition 4.26.

Definition 4.26. Let $\mathcal{O}_i$ be one of the input ontologies and $Sig(\mathcal{O}_i)$ be its signature, let $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ be the integrated ontology, and let A, B be atomic concepts in $Sig(\mathcal{O}_i)$. We define two sets of violations of $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ w.r.t. $\mathcal{O}_i$, with $i = \{1, 2\}$:

- *basic violations*, denoted as $\mathsf{basicViol}(\mathcal{O}_i, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$, as the set of $A \sqsubseteq B$ axioms satisfying:

  (i) $A \sqsubseteq B \in \mathsf{diff}^{\approx}_{Sig(\mathcal{O}_i)}(\mathcal{O}_i, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$,

  (ii) $\mathcal{O}_i \not\models B \sqsubseteq A$, and

  (iii) there is no C in $Sig(\mathcal{O}_i)$ such that $\mathcal{O}_i \models C \sqsubseteq A$, and $\mathcal{O}_i \models C \sqsubseteq B$;

- *equivalence violations*, denoted as $\mathsf{eqViol}(\mathcal{O}_i, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$, as the set of $A \equiv B$ axioms satisfying:

  (i) $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models A \equiv B$,

  (ii) $A \sqsubseteq B \in \mathsf{diff}^{\approx}_{Sig(\mathcal{O}_i)}(\mathcal{O}_i, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ and/or $B \sqsubseteq A \in \mathsf{diff}^{\approx}_{Sig(\mathcal{O}_i)}(\mathcal{O}_i, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$.

△

For basic violations the *assumption of disjointness* can be applied, that is, if two atomic concepts A, B from one of the input ontologies are not involved in a subsumption relationship nor share a common subconcept (excluding $\bot$) they can be considered as disjoint. Hence, the repair of these violations can be reduced to a consistency repair, if the input ontologies are extended with sufficient disjointness axioms. The same does not necessarily hold for equivalence violations, for which a suitable repair algorithm, based on Answer Set Programming, is introduced in Chapter 5. These two repair algorithms are then combined into a single multi-strategy algorithm, presented in Chapter 7, at the basis of our repair tool for conservativity violations.

Note that we assume that the alignment $\mathcal{M}$ is coherent with respect to $\mathcal{O}_1$ and $\mathcal{O}_2$ (that is, $\mathsf{diff}^{\approx}_{\Sigma}(\mathcal{O}^{\emptyset}_{\mathcal{O}_1,\mathcal{O}_2}, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ does not contain any axiom $A \sqsubseteq \bot$, for any $A \in \Sigma = Sig(\mathcal{O}^{\emptyset}_{\mathcal{O}_1,\mathcal{O}_2})$). The main reason for requiring as input a coherent alignment, is that unsatisfiable concepts would be subsumed by any other concept, thus producing a massive number of violations not directly related to the conservativity principle.

Several examples of violations of the conservativity principle are provided in Example 4.10.

Example 4.10. Consider the aligned ontology of Example 4.7, and suppose that axiom $Fish \sqsubseteq \neg Horse$ has been removed from $\mathcal{O}_2$ (in order to have a coherent alignment as input). Let $\Sigma = Sig(\mathcal{O}_2)$.

Axiom $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models SeaHorse_2 \sqsubseteq Horse_2$ belongs to $\mathsf{diff}^{\approx}_{\Sigma}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$, for the interaction between mappings $\langle Horse_1, Horse_2, \equiv, 1\rangle$ and $\langle Horse_1, SeaHorse_2, \sqsupseteq, 0.6\rangle$. Given that no axioms $Horse_2 \sqsubseteq SeaHorse_2$ or $SeaHorse_2 \sqsubseteq Horse_2$ are entailed by $\mathcal{O}_2$, and that these two concepts do not share a common subconcept (excluding $\bot$), this axiom is a basic violation.

Consider, instead, axiom $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models Monkey_2 \sqsubseteq Chimp_2$, that again belongs to $\text{diff}^{\approx}_{\Sigma}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$. Such an axiom is not a basic violation, because $\mathcal{O}_2 \models Chimp_2 \sqsubseteq Monkey_2$ holds. However, we have that $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models Monkey_2 \equiv Chimp_2$, and therefore this axiom is an equivalence violation.

Another violation is represented by axiom $Meat_2 \sqsubseteq Food_2$. However, given that concept $RedMeat_2$ is a common descendant of $Meat_2$ and $Food_2$ (due to the axioms $RedMeat_2 \sqsubseteq Meat_2$ and $RedMeat_2 \sqsubseteq Food_2$), such violation is neither a basic violation, nor an equivalence one. ◊

4.4 Horn Propositional Logic and Answer Set Programming

This section provides the basic notions of Horn Propositional Logic (Section 4.4.1), that will be a key ingredient for achieving scalability for the method proposed in Chapter 6, and Answer Set Programming (Section 4.4.2), that will be at the basis of the repair algorithm of Chapter 5.

4.4.1 Horn Propositional Logic

In the following we provide the definition of Horn propositional rules (Definition 4.27), and Horn propositional formulas (Definition 4.28).

Definition 4.27. Given a set of propositional letters $\boldsymbol{A}$ (the signature), valid *Horn propositional rules* (or *basic Horn formulas*, or simply *clauses*) are defined by the following grammar:

$$\boldsymbol{Body} ::= \top \mid \boldsymbol{A} \mid \boldsymbol{A} \wedge \ldots \wedge \boldsymbol{A}$$

$$\boldsymbol{Head} ::= \bot \mid \boldsymbol{A}$$

$$\boldsymbol{Rule} ::= \boldsymbol{Head} \leftarrow \boldsymbol{Body}$$

In the above grammar $\bot$ and $\top$ represent the Boolean values true and false, respectively. △

The signature of a formula can be omitted when no confusion arises.

Definition 4.28. A *Horn propositional formula* $\mathcal{P}$ is a conjunction of Horn propositional clauses. △

In the remainder we will use the splitted notation for Horn formulas, omitting the conjunctions between clauses, that are then represented as a set.

For Horn propositional logic we have the classic model-based notion of logical entailment of a clause, similarly to what is defined for DL in the previous section. In Definition 4.29 we introduce the notion of satisfiability of a Horn formula.

Definition 4.29. A Horn *propositional formula* $\mathcal{P}$ is said to be *satisfiable* if it is not possible to derive $\bot$, starting from $\top$. $\triangle$

A linear-time algorithm have been proposed in [DG84] by Dowling and Gallier, for testing the satisfiability of Horn propositional formulas (discussed in more details in Section 6.3). Another reasoning service supported by this logic is entailment check of a single proposition a, or of an implication $b \leftarrow a$.

4.4.2 Answer Set Programming

Answer Set Programming[8] (*ASP*) [Fab13, Lif08] is a declarative programming language able to express computationally complex search problems, primarily *NP*-hard ones, but its expressivity covers up to all the problems in the complexity class Σ_2^P, and its complement Π_2^P [Fab13].

We first define the basic notions underlying its syntactical definition.

A *term* is either a variable or a constant. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity n, and $t_1, \ldots, t_n$ are terms. A *classical literal* is either a positive atom p or a negative atom $\neg p$. A *negation as failure* literal l is of the from l or not l (in the former case the literal is called positive, in the latter negative). A set of literals L is called *consistent* if no l and $\neg l$ appear in it at the same time.

ASP programs are (finite) sets of rules of the form $bead \leftarrow body$, where $head$ can be a disjunction of atoms. For this reason, the expressive power of *ASP* rules exceed that of Horn Propositional Logic.[9] Supporting disjunctive formulas in the $head$ makes the formulation of *NP* problems straightforward in *ASP*. More formally, a disjunctive rule is of the form:

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m.$$

Rules with exactly one atom in the $head$ are called *normal rules*, rules without atoms in the $head$ are called *integrity constraints*, while rules with $head$ having strictly more than one atom are called *generative rules*. Rules with an empty $body$ (*i.e.*, $k = m = 0$) are called *facts*.

Given a rule r as above, we denote the set of literals in its $head$ as $H(r) = \{a_1, \ldots, a_n\}$, while we denote with $B(r) = B^+(r) \cup B^-(r)$ those in its $body$, where $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$, which are called the positive $body$ and the negative $body$, respectively.

Integrity constraints prune the search space excluding the identified solutions that do not satisfy the constraints, normal rules infer new consequences from the known facts (original facts or de-

[8] The content of this section is adapted from [Fab13].

[9] Even if some forms of disjunctive rules can be reduced to a "shifted-version", where disjunction in the head is not strictly necessary and can be emulated by means of multiple rule bodies of (unstratified) normal rules [Fab13]. However this is, in general, not always possible.

rived ones), and generative rules fork the generation of a number of different solutions depending on the atoms in the $head$ of the rule. This reasoning methodology is described as "guess-and-check" approach.

ASP has also been extended with optimization-oriented constructs, that are used in rules requiring the minimization of a numeric variable or the cardinality of a predicate, and are usually referred to as *soft constraints* (in constrast to integrity constraints that are called *hard constraints*).

The solutions (possibly none) to the problem encoded by an *ASP* program are called *stable models*, and given that they are usually sets, the term *answer sets* has been introduced. A program having no stable models is called *unsatisfiable*.

In order to formally define the semantics of (positive disjunctive) *ASP* programs, we need to introduce some preliminary notions.

The *Herbrand universe* of a program Π, denoted as U_Π, is defined as the set of constants appearing in Π. A term (atom, rule, program) is called *ground* if no variables appear in it. The *Herbrand literal base*, denoted as B_Π, is the set of all ground literals constructable from predicate symbols appearing in Π. The grounding of a given rule r of a program Π, denoted as $Ground(r)$, is equivalent to the set of rules obtained by replacing each variable in r by constants in U_Π, in all possible ways. Similarly, the ground instantion of an *ASP* program Π is defined as $Ground(\Pi) = \bigcup_{r \in \Pi} Ground(r)$.

An *interpretation* is a consistent set of ground classical literals $I \subseteq B_\Pi$ w.r.t. a program Π. A consistent interpretation is called *closed under* Π if, for every $r \in Ground(\Pi)$, $H(r) \cap X \neq \emptyset$ whenever $B(r) \subseteq X$. An interpretation $X \subseteq B_\Pi$ is an answer set for a positive disjunctive program Π if it is minimal, under set inclusion, among all the consistent interpretations that are closed under Π.

The *reduct* or *Gelfond-Lifschitz transform* of a ground program Π w.r.t. a set $X \subseteq B_\Pi$ is the positive ground program Π^X, obtained from Π by:

- deleting all rules $r \in \Pi$ for which $B^-(r) \cap X \neq \emptyset$ holds,
- deleting the negative body from the remaining rules.

An answer set of a program Π is a set $X \subseteq B_\Pi$ such that X is an answer set of $Ground(\Pi)^X$.

4.5 Graph-Theory

This section provides the necessary preliminaries about Graph-Theory.

Definition 4.30 and Definition 4.31 introduce the definitions of *directed graph* and of its weighted counterpart, respectively.

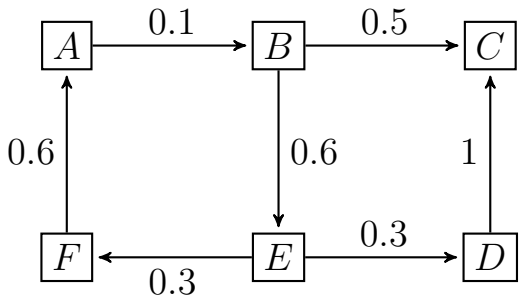


Figure 4.2: A simple example of (weighted) digraph.

Definition 4.30. A *directed graph* (digraph) G is a set of vertices V together with an antireflexive relation A on V. △

Definition 4.31. A *weighted digraph* G is a digraph where each arc has a third component, called *weight* of the arc, assuming values in an appropriate structure (*e.g.*, $\mathbb{N}, \mathbb{R}$, etc.). Given an arc $a = (u, v, c)$, we denote its weight as $w(a) = c$. Function w can also be applied to a set of arcs, representing the sum of the weights of its elements. △

In the remainder of the thesis, if not explictly stated, we always refer to *weighted digraphs*, even when the general terms *graph* or *digraph* are employed. For presentation purposes, when one of the components of an arc is not relevant, it may be replaced by the placeholder "_", representing any legal value that the considered component may assume.

An instance of digraph and weighted digraph is given in Example 4.11.

Example 4.11. Consider the graph G, composed by vertices $V = \{A, B, C, D, E, F\}$, and (weighted) arcs $A = \{(A, B, 0.1), (B, C, 0.5), (B, E, 0.6), (D, C, 1), (E, D, 0.3), (E, F, 0.3), (F, A, 0.6)\}$, and shown in Figure 4.2. If we ignore the third component of each arc, G is an example of digraph, while considering it turns G into a weighted digraph. ◊

Definition 4.32 and Definition 4.33, instead, formalize the notion of *subgraph* and *induced subgraph* for directed graphs, respectively.

Definition 4.32. Given a digraph $G = (V, A)$, a *subgraph* $G' = (V', A')$ of G is a digraph such that $V' \subseteq V$, and $A' \subseteq A$, and for any arc $a = (u, v, \_) \in A'$, we have that $u, v \in V'$. △

Definition 4.33. Given a digraph $G = (V, A)$, a subgraph $G' = (V', A')$ of G is an *induced subgraph* of G, denoted as $G|_{G'}$, if, for any pair of vertices $u, v \in V'$, $(u, v, \_) \in A'$ iff $(u, v, \_) \in A$. In other words, G' is an induced subgraph of G if it has exactly the arcs that appear in G, over a subset of the vertex set. G' can be also denoted as $G|_{V'}$ and is said to be induced by V'. △

An instance of subgraph and induced subgraph for a digraph is given in Example 4.12.

Example 4.12. Consider the (weighted) digraph G of Example 4.11. $G' = (V', A')$ is a subgraph of G, where $V' = \{A, B, C\}$ and $A' = \{(A, B, 0.1), (B, C, 0.5)\}$. G' is also the induced subgraph of G w.r.t. the set of vertices composing V'. If either of the two arcs of A' is missing, G' would still be a subgraph, but not an induced subgraph. $\Diamond$

The classic notion of *path* and *cycle* are given in Definition 4.34 and Definition 4.35, respectively.

Definition 4.34. Given a digraph G, a (directed) *path* $\pi = [v_1, \ldots, v_n]$ of G, with $n > 1$, is a sequence of vertices where each pair of vertices v_i and v_{i+1}, with $i \in 1 \ldots n-1$, is connected by an arc $(v_i, v_{i+1}, \_) \in A$. The *length* of such a path π is $n-1$, denoted with $length(\pi)$. Given two vertices $u, v \in V$, u is *reachable* from v iff a path π of length $m-1$ exists such that $v_1 = u$ and $v_m = v$, for some $m > 2$. $\triangle$

Definition 4.35. Given a digraph G and a directed path $\pi = [v_1, \ldots, v_n]$ of G, we define it a directed cyclic path (*cycle* in what follows) iff the first and last vertices coincide, *i.e.*, $v_1 = v_n$. We say that a cycle κ is *broken* by the removal of any of its arcs (resp. vertices). We also say that a set of arcs (resp. vertices) breaks a set of cycles iff any of the cycles contains at least one element of the set. Note that we do not consider self-arcs, and therefore cycles with length equal to 1 are not allowed. Given a digraph G and two cycles of G, we define them *distinct cycles* if they are not a cyclic permutation one of the other. $\triangle$

In Example 4.13, an instance of path and cycle is given.

Example 4.13. Consider the (weighted) digraph G of Example 4.11. Vertex D is said to be reachable from vertex A, due to the existence of the path $[A, B, E, D]$, having length 3. $[A, B, E, F, A]$ is, instead, a cycle. $\Diamond$

Given an arc $a = (u, v, \_)$, we refer to $\mathcal{V}(a) = \{u, v\}$ as the set containing its source and target vertices. $\mathcal{V}$ naturally extends to sets (resp. sequences) of arcs, as the union of its application on each element of the set (resp. sequence).

Given a path $\pi = [v_1 \ldots v_n]$, we refer to $\mathcal{A}(\pi) = \{(v_i, v_{i+1}, \_) \mid i \in [1 \ldots n-1]\}$ as the set containing all of the arcs of π. Similarly, given a set of vertices V' such that a digraph $G = (V, A)$ exists and $V' \subseteq V$, we refer to $\mathcal{A}(V') = \{(u, v, \_) \in A \mid u, v \in V'\}$ as the set containing all the arcs of G between vertices of V'.

After the introduction of the notion of vertex reachability, we are now ready to give the definitions of *strongly connected component* (Definition 4.36), and *strongly connected graph*, (Definition 4.37).

Definition 4.36. Given a digraph $G = (V, A)$, a *strongly connected component* (SCC) of G is a maximal set of vertices $C \subseteq V$ such that for all $u, v \in C$, both u is reachable from v and

viceversa. Notice that at least a cycle containing all the elements of the SCC exists, so cycle detection in a digraph G can be reduced to the identification of the SCCs of G. The set of the SCCs of a digraph G is denoted with $SCC(G)$. $\triangle$

Definition 4.37. Given a digraph $G = (V, A)$, if $SCC(G) = \{S\}$, and $S = V$, then G is a *strongly connected digraph*. $\triangle$

In Example 4.14, an example of each of the aforementioned notions is given.

Example 4.14. Consider the (weighted) digraph G of Example 4.11. The set of vertices $V' = \{A, B, E, F\}$ is a strongly connected component of G. The induced subgraph $G|_{V'}$ is a strongly connected digraph. $\Diamond$

Tarjan's Algorithm and *Johnson's Algorithm*, related respectively to SCCs and cycles, are introduced in Definition 4.38 and Definition 4.39, respectively.

Definition 4.38. *Tarjan's Algorithm* [Tar72] (Tarjan) finds the SCCs of a digraph $G = (V, A)$ using a single depth-first search with a time complexity in $\mathcal{O}(|V| + |A|)$. $\triangle$

Definition 4.39. *Johnson's Algorithm* [Joh75] (Johnson) computes all the distinct cycles in a digraph $G = (V, A)$ with time complexity in $\mathcal{O}((|V| + |A|) \cdot (c + 1))$ and space complexity in $\mathcal{O}(|V| + |A|)$, where c is the number of distinct cycles in G. $\triangle$

Feedback Edge Set (and related weighted formulation), a fundamental problem related to cycles in digraphs, is discussed in Definition 4.40.

Definition 4.40. Given a digraph $G = (V, A)$ with cycles, the *Feedback Edge Set* (*FES*) problem aims at selecting a subset of A, called *feedback (edge) set*, with minimum cardinality, whose removal makes G acyclic. This problem is known to be *NP*-hard [ENSS98], as well as its weighted variant, *Weighted Feedback Edge Set* (*WFES*). *WFES* differs from *FES* for what concerns the minimization objective for the feedback set, that is required to be minimal w.r.t. the sum of the weights of its elements. *FES* problem is also equivalent to *Feedback Vertex Set* (*FVS*) problem [ENSS98], where a subset of V that makes G acyclic is sought. $\triangle$

An instance of the aforementioned algorithms and problem is provided by Example 4.15.

Example 4.15. Consider the (weighted) digraph G of Example 4.11. Tarjan's Algorithm, given as input G, would produce as output the following set of SCCs: $\{\{A, B, E, F\}, \{C\}, \{D\}\}$. Johnson's Algorithm, given as input G, would identify $[A, B, E, F, A]$ as the only cycle in the graph. A solution for *FES* over G would be any singleton subset of the following set of arcs $\{(A, B, 0.1), (B, E, 0.6), (E, F, 0.3), (F, A, 0.6)\}$. Instead, the solution of *WFES* over G is $\{(A, B, 0.1)\}$. The solution of *FVS* is any subset of $\{A, B, E, F\}$ that is a singleton. $\Diamond$

4.6 State of the Art

This section provides a general overview of the state of the art on ontology alignment evolution (Section 4.6.1), the general ontology debugging techniques (indirectly) related to the conservativity principle (Section 4.6.3), and an overview over the relevant approaches for ontology alignment repair (Section 4.6.4).

As already discussed in Section 4.3.3, the theoretical foundation of the conservativity principle is represented by the notion of conservative extension. Section 4.6.2 discusses the related work on the subject and provides an overview of the decidability and complexity results. The computational hardness justifies the interest in approximated notions and repair techniques.

Detailed comparisons with specific approaches tightly related to the addressed problems are postponed to the relevant chapter.

4.6.1 Ontology Alignment Evolution

Martins *et al.* [MS09] propose an automatic ontology mapping reformulation algorithm based on update logs, that cannot be applied when at least one of the evolving ontologies is externally updated, and no assumptions can be made on the update management. In scenarios where change logs are not available, an automatic ontology mapping reformulation algorithm is applied (user intervention is required only when the adaptation leads to an inconsistent ontology). This automatic algorithm consists of a set of adaptation rules exploiting the transitivity property of the subsumption relation.

The approach by Khattak *et al.* [KPL+11] proposes an automatic adaptation technique of existing mappings in response to ontology update, and is based on a specific change log format, enriched with temporal and provenance information. This proposal shares the same limitations of the approach of Martins *et al.* based on change logs.

Dos Reis *et al.* [DRPDSRD12] propose general guidelines and requirements for a heuristic-based framework for knowledge base alignments evolution. A concrete set of change operations for implementing mapping adaptation has been defined in [DRDP+13], on top of the ontology change primitive operations supported by the ontology differencing algorithm of Hartung *et al.* [HGR10].

Stemming from [DRDP+13], in [GDRH+13] a multi-strategy approach for ontology mapping adaptation in response to ontology changes is proposed. The ontology delta is computed using the algorithm of Hartung *et al.* [HGR10] and, for each concept deletion, it reformulates the subsumption axioms involving the deleted concept.

For semantic mappings between XML schemas and ontologies a proposal by An *et al.* [ABM08]

exists. This work proposes a (semi-)automatic adaptation of existing mappings in response to XML schema evolution. The proposed stategies exploit several characteristics of XML schemas (*e.g.*, the tree structure of the described XML documents), thus the approach is not directly applicable in the context of ontology mappings.

4.6.2 Conservative Extension

As already discussed in Section 4.3.4, the general formulation of the conservativity principle, tackled in this second part of the thesis, is known as conservative extension.

As introduced in Section 4.3.2, ontology mappings are frequently interpreted as (subsumption or equivalence) axioms in the aligned ontology, and the detection of conservativity principle violations (w.r.t. the consequence relation, as formulated in [JRCHB11a]) reduces to checking whether the aligned ontology is a conservative extension of each input ontology.

Unfortunately, as discussed in Section 4.3.3, testing for conservativity is *EXPTIME*-complete for simple DLs such as $\mathcal{EL}$ [LW10], *2ExpTime*-complete for $\mathcal{ALC}$, $\mathcal{ALCQI}$ and $\mathcal{ALCQI}$, and it is already undecidable for DLs at least as expressive as $\mathcal{ALCQIO}$ [LWW07]. The prohibitive computational complexity justifies the interest in heuristic and approximate methods.

The notion of conservative extension has also been used to define modules. Given an ontology $\mathcal{O}$, a module $\mathcal{O}'$ is a subset of $\mathcal{O}$, such that $\mathcal{O}$ is a conservative extension of $\mathcal{O}'$ w.r.t. its signature [CHKS07].

Another application of interest for conservative extension is ontology versioning (in particular, semantic differencing) where it is claimed that one of the most interesting impact of changes is the effect on the ontology classification and named-concept hierarchy [GPS12].

As already discussed, the tight relationship between ontology development, change management and debugging techniques clearly emerges in systems incorporating all these features and exploiting their mutual effects, as [JRCHB11b].

4.6.3 Conservativity Principle, Ontology vs Alignment Repair

Once a violation of the conservativity principle is detected, there are different approaches to correct it by modifying the aligned ontology. These repairs can then target the input ontologies or the ontology alignment.

The first approach, similar to that of Section 4.3.5 for mapping incoherence, is to consider as problematic all the violations, and to correct them using classic ontology repair strategies on the aligned ontology. An example is to compute the set of all the justifications [KPHS07, HPS08, HPS10] for the unwanted axiom, and to select an hitting set (of axioms) to be removed, with

different possible minimality criteria (*e.g.*, the number of removed axioms) [Sch05]. Given that the aligned ontology is unaware of the provenance of the different entities and axioms, the hitting set may remove elements either from the ontologies, or from the alignment, or a mixture of the two.

The second one is a family of approaches from Lambrix *et al.* [LWKDI13, LDI13], which targets violations that can be considered as false positives, for which the problem source is considered to stem from the incompleteness of the input ontologies. The correction strategy aims at inserting, between different alternatives, the minimal set of axioms needed in order to obtain the novel axiom as a consequence in the input ontology too (solving, in this way, the violation).

A unified approach [IL13, LL13] has been proposed. For each detected violation different repair plans are generated, and they are ranked w.r.t. their informativity (concept that captures the conciseness of the repair by considering the number of solved depending violations) and minimality (based on the number of inserted/deleted axioms). It is always possible for the user to classify the violations as false or true positives, and to influence in this way the plan ranking. Our heuristic could be conceived as an alternative and additional plan in this multistrategy approach, that supports both automatic and assisted repair.

[IL13, LL13, LWKDI13] refer to the same (simplified) context addressed here in Chapter 5, namely taxonomies and taxonomical skeleton of more expressive DLs, claiming that this feature is one of the most used in semantic-enabled applications, that often do not exploit the more advanced features provided by expressive DLs.

As already discussed, our heuristic is orthogonal to the others discussed in this section, and could be an alternative inside a multistrategy approach. However, there are settings for which the only repair target is the alignment (refer to Section 4.7 for more details).

Another aspect of interest is the ranking criteria used by multistrategy approaches. Our repair computation relies on a minimality measure expressed in terms of the total sum of confidence of the removed mappings. For scenarios in which this confidence is missing or unreliable, it is possible to rank and select among possible repairs (and possibly among that of other alternative strategies), or to try to estimate or correct the confidence values. An example of an algorithm for this approach is discussed in [JRCHB11a], where a locality-based heuristic is proposed for estimating missing confidence values.

4.6.4 Repair Techniques Targeting Ontology Alignments

In this section we discuss the related work dealing with the detection and correction of violations of the different principles, that ontology mappings should satisfy. All the techniques discussed in this section target mappings only, and consider the input ontologies as not modifiable.

ALCOMO. ALCOMO [Mei11] is a tool devoted to the detection and correction of incoherent ontology mappings. It is sound and complete up to $\mathcal{SHIN}(\boldsymbol{D})$, and it aims at computing a diagnosis, that is, a set of mappings whose removal from the original alignment restores satisfiability in the aligned ontology. Good performance is achieved be means of approximate strategies that aims at minimizing the reasoning in the aligned ontology, while computing a global diagnosis for possibly non disjoint sets of problematic mappings.

In what follows we analyze the repair features of ASMOV, Lily, YAM++, LogMap and AML matchers (summarized in Table 4.3), that address also problematic patterns not violating the consistency principle.

ASMOV. ASMOV matcher [JMSK09] supports five kind of problematic patterns (shown in Figure 4.4 and summarized in Table 4.3):

(a.i) multiple-entity mappings,

(a.ii) criss-cross mappings,

(a.iii) disjointness-subsumption contradiction,

(a.iv) subsumption incompleteness,

(a.v) domain and range incompleteness.

(a.iii) is related to the consistency principle (it is a subcase of what is considered by ALCOMO), (a.iv) and (a.v) are related to the locality principle, and deal with alignment incompleteness, that is out of the scope of the thesis. (a.i) and (a.ii) always fall into the conservativity principle when "$\equiv$" semantic relation is employed, and this partially holds for (a.ii) when "$\sqsubseteq$" and "$\sqsupseteq$" semantic relations are used, but only with a particular combination of mappings (see Figure 4.3).

Lily. Lily matcher [WX12] detects:

(b.i) redundant mappings,

(b.ii) multiple-entity mappings,

(b.iii) "$\sqsubseteq$-cycle",

(b.iv) disjointness-subsumption contradiction,

(b.v) abnormal-mappings (that is, mappings violating the locality principle).

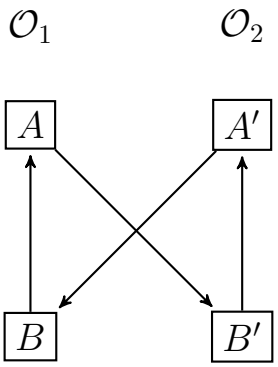


Figure 4.3: An example of equivalence violation employing "$\sqsubseteq$" semantic relation.

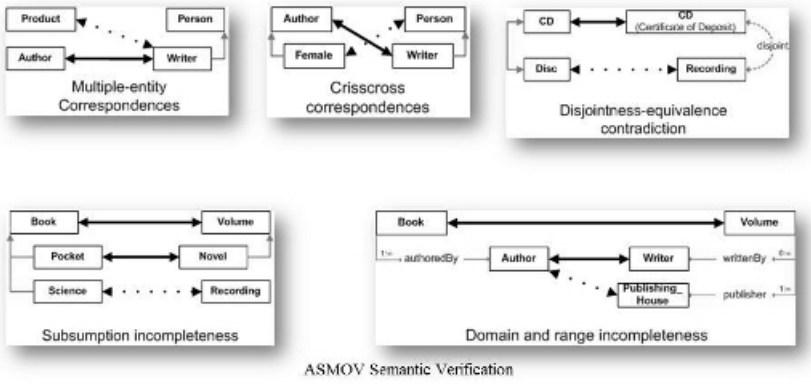


Figure 4.4: Problematic mapping patterns detected by ASMOV (source `http://infotechsoft.com/products/asmov.aspx`).

Problem	Description	Violated Principle	Notes
ASMOV			
(a.i)	multiple-entity mappings	*Conservativity*	only with $\equiv$-relation
(a.ii)	criss-cross mappings	*Conservativity*	special case of (b.iii)
(a.iii)	disjointness-subsumption conflict	*Consistency*	special case of ALCOMO
(a.iv)	subsumption incompletness	–	–
(a.v)	domain/range incompletness	–	–
Lily			
(b.i)	redundant mappings	–	–
(b.ii)	multiple-entity mappings	*Conservativity*	same as (a.i)
(b.iii)	subsumption cycle	*Conservativity*	–
(b.iv)	disjointness-subsumption conflict	*Consistency*	same as (a.iii)
(b.v)	abnormal-mappings	*Locality*	–
YAM++			
(c.i)	disjointness-subsumption conflict	*Consistency*	same as (a.iii)
(c.ii)	criss-cross mappings	*Conservativity*	same as (a.ii)
(c.iii)	property-property mapping conflict	–	same as (a.v)
(c.iv)	duplicated conflict	*Consistency*	same as (a.i)
LogMap			
(d.i)	multiple-entity mappings	*Conservativity*	same as (a.i)
(d.ii)	criss-cross mappings	*Conservativity*	same as (a.ii)
(d.iii)	consistency principle	*Consistency*	special case of ALCOMO
(d.iv)	subsumption incompletness	–	same as (a.iv)
(d.v)	domain/range incompletness	–	same as (a.v)
(d.vi)	locality principle	*Locality*	similar to (b.v)
AML			
(e.i)	consistency principle	*Consistency*	special case of ALCOMO
(e.ii)	domain/range incompletness	–	same as (a.v)

Table 4.3: Problematic patterns supported by ASMOV, Lily, YAM++, LogMap, and AML ontology matchers.

(b.i) and (b.v) are out of scope, because not related to conservativity principle. (b.iii) captures one of the possible patterns resulting into an equivalence violation (similar to what addressed in Chapter 5). For (b.ii) the same considerations made for (a.i) and (a.ii) apply.

In our opinion, multiple mappings cannot be considered as a problem *per se*, but only when they are responsible for a violation of some principle, such as the conservativity principle: for instance, multiple mappings with different equivalent concepts, besides being redundant from a logical view-point, could be informative if the equivalence is entailed but not explictly stated with a direct axiom in the input ontology. Whenever one of these problems is detected, Lily raises

an error/warning, but its repair is delegated to the user. This option is not always viable due to the potentially massive size of the involved ontologies/alignments, and to the difficulty of manually detecting diagnoses/repairs. Manual repair seems even less feasible, given the broad range of ontology and ontology mappings users, that most of the time are not computer scientists and are not familiar with this kind of formalisms and problems. No details are provided on problematic mapping detection techniques employed by Lily.

YAM++. YAM++ matcher [NB12, NB13] supports four problematic patterns:

(c.i) disjointness-subsumption conflict (identical to (a.iii)),

(c.ii) criss-cross mappings (identical to (a.ii)),

(c.iii) property-property mapping conflict (identical to (a.iv)),

(c.iv) duplicated conflict (identical to (a.i)).

LogMap. LogMap [JRC11] is an ontology matcher with logic-based repair features, that is sound but not complete. It encodes the mappings into *Horn Rules*, and implements a variant of the linear-time *Dowling-Gallier* algorithm [DG84] for propositional *Horn* satisfiability. LogMap repair facility is also employed by ServOMap ontology matcher [BD13]. LogMap also employs patterns (a.i), (a.ii), (a.iv) and (a.v), but they are used to reduce the mappings confidence, and not to remove them. Summarizing, the supported patterns are the followings:

(d.i) multiple-entity mappings (identical to (a.i)),

(d.ii) criss-cross mappings (identical to (a.ii)),

(d.iii) consistency principle,

(d.iv) subsumption incompleteness (identical to (a.iv)),

(d.v) domain/range incompleteness (identical to (a.v)),

(d.vi) locality principle (similar to (b.v)).

AML. AML matcher [SFPC13] is equipped with a repair algorithm that minimizes both the incoherence of the resulting alignment and the number of removed mappings. Scalability is achieved using heuristics to determine near-optimal solutions and a modularization-based technique that extracts only the relevant fragments of the ontologies needed for incoherence repair. Additionally, AML also prevents mappings between properties violating pattern (a.v). Therefore, the patterns supported by AML can be summarized as follows:

(e.i) consistency principle,

(e.ii) domain/range incompleteness (identical to (a.v)).

The experimental analysis of the impact of the repair and detection patterns of ASMOV and Lily has been restricted to equivalence violations only (see Section 5.5.4), given that they (directly or indirectly) address only this kind of violations. Instead, the analysis of AML, LogMap and YAM++, given their participation until the 2014 (2013 for YAM++) edition of *OAEI*, has been covered for both equivalence and subsumption violations, because their alignments were included in the considered dataset (*OAEI* 2012–2014).

4.7 Motivating Scenarios

This section introduces different scenarios motivating the work addressed in the second part of the thesis, related to the conservativity principle. Specifically, Section 4.7.1 addresses the role of conservativity principle in Multi-Agent Systems, Section 4.7.2 the consequences of conservativity principle violations in query answering scenarios, while Section 4.7.3 investigates the interplay between ontology alignment repair and ontology alignment evolution.

By the analysis of the literature, we were able to derive different perspectives from which the problem could be tackled. The hypotheses behind these alternative views are substantially three:

(Hyp.i) conservativity principle violations can be false positives stemming from the possible incompleteness of the input ontologies;

(Hyp.ii) the repair for a conservativity principle violation could include axiom addition/removal in the input ontologies;

(Hyp.iii) conservativity principle does not affect tasks based on ontology alignments.

(Hyp.i) and (Hyp.ii) share a common scenario, in which a false positive is addressed by inserting a sufficient number of axioms in order to allow the interested input ontology to entail the violation (and solving, in this way, the violation itself). This family of approaches from Lambrix *et al.* has been discussed in Section 4.6, and we briefly recap the main ingredients. For each detected violation, different repair plans are generated, that are ranked w.r.t. their informativity and minimality, and it is always possible, for the user, to classify the violations as false or true positives.

(Hyp.i) is only sensible if we apply our techniques in a completely automatic way. In this setting, however, the aforementioned approaches are equally problematic, given that no techniques are

available for automatically discriminating between true and false positives. In addition, a real-world assessment in the biomedical domain [BH12] confirms that conservativity violation false and true positives are equiprobable, and that therefore none of the two options can be exclusively considered and automatically applied. Our approach, for instance, could be conceived as an alternative and additional plan in the multistrategy approach of [IL13, LL13], which supports both automatic and assisted repair. Therefore, even if the techniques we propose are described and experimented in a completely automatic setting, they can be integrated into an interactive system supporting user involvement (an example are the GUIs shown in Sections 5.5.9 and 6.5.5).

Concerning (Hyp.ii), there are settings for which the strong constraint of having sealed input ontologies is not optional, and the other strategies are not applicable. In this settings, despite the input ontologies can be incomplete, they are not modifiable by their end-user (human or not). The main examples come from ontology alignment repair for ontology matchers (consider [Mei11, AR13, JRC11, JRCHB11a]), where the input ontologies are meant as immutable, and where any repair for the detected defects can only be expressed as a subset of the given alignment. Section 4.7.1, instead, provides reference scenarios in Multi-Agent Systems area.

It is evident that the aforementioned hypothesis is not applicable to the detection techniques, given that they are independent from the repair technique and would be needed in any case. (Hyp.iii), instead, affects detection techniques as well, and it is addressed in Section 4.7.2.

4.7.1 Multi-Agent Systems

In the context of Multi-Agent Systems we have an example of scenario that could require conservativity principle (detection and/or repair) techniques.

The first example is provided by the author of [MABR11, MAB$^+$14], that consider the extension of a DL-based agent language with agent-to-agent plan exchange based on the alignment computed by an ontology matcher. In such a context, a *sceptical* approach (opposed to *optimistic*) requires that highly critical tasks are not performed if violations in the aligned ontology are detected, as mentioned by the author in [MAB$^+$14]. In this approach the input ontologies are considered as immutable, and therefore the approach of Lambrix *et al.* could not be applied.

Another approach, described in [PT14], proposes a novel inquiry dialogue (*i.e.*, a dialogue between two entities with the aim of solving a goal through knowledge exchange) that aims at reusing existing mappings related to a domain of interest, with a minimal disclosure of private knowledge. Each agent is equipped with a DL local (private) ontology.

In this context on one hand, the ontology of the other agent(s) cannot be altered, on the second hand, modifying the local ontology after any dialogue could pose some risks: consider the effect of a malicious agent forging ad-hoc mappings with the aim of altering the local ontology of the interlocutor. Different policies could be conceived for the modification of the local ontology, the

approach of Lambrix *et al.* could be referred to as optimistic, while forbidding alterations to the local ontology as sceptical. Of course mixed approaches could be conceived as well.

The sceptical approach could also use consistency/conservativity/locality principle detection techniques in order to estimate, under a game-theoretic point of view, the convenience of dealing with a particular agent, given the effort needed to soften the edges (resp. risk) represented by these violation(s). In case of too many violations, the agent could simply decide to refuse to communicate, and seek for another (hopefully more compatible) agent.

4.7.2 Query Answering

OAEI introduced in 2014 the *OA4QA* track [SJP14, DEE+14] in order to investigate the consequences of performing query anwering in presence of violations affecting ontology alignments (including those related to conservativity principle), and to evaluate the effectiveness of the repair strategies employed by the matchers.

Violations in ontology alignments are considered as a possible threat for ontology alignment usefulness [Mei11, JRCHB11a], but their practical effect is, however, clearly evident only when alignments are involved in complex tasks such as query answering [Mei11]. The traditional tracks of OAEI evaluate ontology matching systems w.r.t. scalability, multi-lingual support, instance matching, reuse of background knowledge, etc. Systems' effectiveness is, however, only assessed by means of classical information retrieval metrics (*i.e.*, precision, recall and f-measure) w.r.t. a manually-curated reference alignment, provided by the organizers.

OA4QA track evaluates those same metrics, w.r.t. the ability of the generated alignments to enable the answering of a set of queries in an OBDA scenario, where several ontologies exist. The target OBDA scenario presents one ontology providing the vocabulary to formulate the queries (QF-Ontology) and a second one linked to the data, that is not visible to the users (DB-Ontology).

Such OBDA scenario is a simplification of real-world use cases, like the Optique project[10] [KJRZ+13, SJRG14, GHJR+15]. Optique advocates for an OBDA approach so that end-users, instead of directly composing queries against the database, formulate queries by means of the vocabulary of a domain ontology (corresponding to the QF-Ontology), that acts as a conceptual abstraction of the data of interest. Ontology entities therefore need to be linked to the database through *ontology-to-schema* mappings.[11] However, in order to be loosely coupled with such domain ontology, the ontology-to-schema mappings are specified against an ontology directly boostrapped from the database (corresponding to the DB-Ontology). Given that only the vocabulary of the bootstrapped ontology is directly linked to the database, we need to compute an alignment between the two involved ontologies. Ontology-based queries (*e.g.*, SPARQL

[10] `http://www.optique-project.eu/`

[11] Ontology-to-schema mappings are out of the scope of the thesis, the interested reader can refer to [CDL+09].

queries) are then automatically rewritten to SQL and executed over the database, exploiting both ontology-to-ontology and ontology-to-schema mappings.

Impact of the Mappings in the Query Results

The impact of unsatisfiable ontologies, related to the consistency principle, is immediate. The conservativity principle, compared to the consistency principle, received less attention in literature, and its effects in a query answering process is probably less known.

In the context of the *conference* dataset of *OAEI*, consider the aligned ontology $\mathcal{O}^{\mathcal{M}}$ computed using *confOf* and *ekaw* as input ontologies ($\mathcal{O}_{confOf}$ and $\mathcal{O}_{ekaw}$, respectively), and the reference alignment between them. $\mathcal{O}^{\mathcal{M}}$ entails $ekaw{:}Student \sqsubseteq ekaw{:}Conf\_Participant$, while $\mathcal{O}_{ekaw}$ does not, and therefore this represents a conservativity principle violation. Clearly, the result set for the query $q(x) \leftarrow ekaw{:}Conf\_Participant(x)$ will erroneously contain any student not actually participating at the conference. The explanation for this entailment in $\mathcal{O}^{\mathcal{M}}$ is given below, where Axioms 4.1 and 4.3 are mappings from the reference alignment.

$$confOf{:}Scholar \equiv ekaw{:}Student \tag{4.1}$$

$$confOf{:}Scholar \sqsubseteq confOf{:}Participant \tag{4.2}$$

$$confOf{:}Participant \equiv ekaw{:}Conf\_Participant \tag{4.3}$$

Possible (minimal) alignment repairs for the aforementioned violation are the following:

- the softening of Axiom 4.1 into $confOf{:}Scholar \sqsupseteq ekaw{:}Student$,
- the weakening of Axiom 4.3 into $confOf{:}Participant \sqsupseteq ekaw{:}Conf\_Participant$.

Repair strategies could disregard weakening in favor of complete mapping removal, in this case the removal of either Axiom 4.1 or Axiom 4.3 would be possible repairs. Finally, for strategies including the input ontologies as possible repair target, the removal of Axiom 4.2 can be proposed as a legal solution to the problem, as well as the addition of axiom $ekaw{:}Student \sqsubseteq ekaw{:}Conf\_Participant$ to $\mathcal{O}_{ekaw}$.

Reflections

Alignment repair does not only affect precision and recall while comparing the computed alignment w.r.t. a reference alignment, but it can enable or prevent the capability of an alignment to be used in a query answering scenario. This conflicting effect in the process of query answering imposes a deeper reflection on the role of ontology alignment repair strategies, depending on the target scenario, similarly to what discussed in [PFSC13] for incoherence alignment repair.

The detailed discussion of the results of the evaluation are out of the aims of the chapter, and are provided in Appendix B. What clearly emerged from *OA4QA* is that logical violations (both consistency and conservativity) play a major role in query answering tasks, and this is therefore a counterargument for (Hyp.iii) (discussed in Section 4.7).

4.7.3 Ontology Alignment Evolution

As already discussed in Section 1.1, logic-based formalisms require not to disregard semantics while considering evolutionary aspects, and our hypothesis is that ontology alignments are not an exception. Given the relevance of ontology alignment adaptation in the context of change management, the core topic of the thesis, we will analyze the related motivating scenario in greater details than for the other scenarios.

Specifically, in order to validate our hypothesis, we analyze all the adaptation strategies of the main contributions to the field of ontology alignment adaptation [MS09, GDRH$^+$13]. None of these strategies consider logical consequences during alignments adaptation, and we therefore aim at highlighting the drawbacks of purely syntactical change management techniques, in the context of metadata equipped with a formal semantics. In addition, we contextually suggest how detection and repair techniques for violations affecting ontology mappings could be integrated in these techniques.

We first consider the approach by Martins *et al.* [MS09]. Assume that $\mathcal{M}$ is a mapping between ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$. The change model reacts to concept deletions, and supports the following adaptation strategies for mappings of the form $m_1 = \langle A, A', \sqsubseteq, \_ \rangle$, with $A \in N_C(\mathcal{O}_1)$, and concept $A' \in N_C(\mathcal{O}_2)$ subject to deletion:

ar.(i) replace m_1 with $\langle A, \top_2, \sqsubseteq, \_ \rangle$, where $\top_2$ is the top concept of $\mathcal{O}_2$ (supposed to be distinct from that of $\mathcal{O}_1$),

ar.(ii) replace m_1 with $\langle A, C', \sqsubseteq, \_ \rangle$, with C' any concept in $N_C(\mathcal{O}_2)$ such that $\mathcal{O}_2 \models A' \sqsubseteq C'$,

ar.(iii) delete m_1.

Obviously, adaptation rules (ar.i) and (ar.iii) cannot preserve conservativity violations requiring mapping m_1, differently from (ar.ii). This directly follows from the observation that (ar.ii) is the only knowledge preserving adaptation rule, among the proposed ones. In order to undestand the possible effects of rule (ar.ii), we need to consider the different scenarios in which the rule can be applied.

If mapping m_1 is not involved in a conservativity violation, it is evident that no new violations can arise from the application of rule (ar.ii), independently from the concrete ontologies and

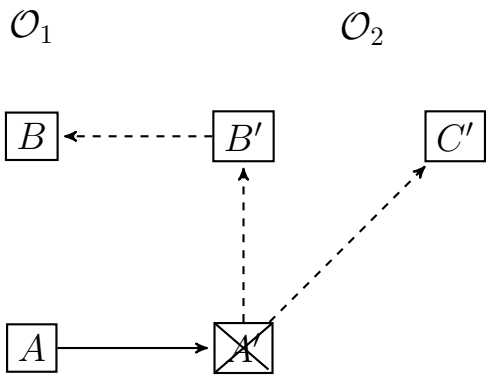


Figure 4.5: An example of application of rule (ar.ii). Dashed arcs represent possibly implicit knowledge.

mappings we consider. However, suppose that a violation $A \sqsubseteq B$ exists in the aligned ontology $\mathcal{O}^{\mathcal{M}}$, stemming from $\mathcal{M}$ and the input ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, where $A, B \in N_C(\mathcal{O}_1)$, as represented in Figure 4.5. In this situation, the choice of the superclass for the mapping adaptation is crucial. Both B' and C' are legal candidates for rule (ar.ii), but while using B' will preserve the conservativity violation, choosing C' would suppress it.

From this consideration it is possible to derive a modified version of the rule, consisting in the following steps for the deletion of concept A':

1. Detects the conservativity violations in $\mathcal{O}^{\mathcal{M}}$
2. For each mapping $\langle A, A', \sqsubseteq, \_ \rangle$:
   - if no violation $A \sqsubseteq B$ exists, apply the normal strategy;
   - otherwise, if a violation $A \sqsubseteq B$ exists, perform adaptation using any concept A'' such that $\mathcal{O}_2 \models A' \sqsubseteq A''$ and $\mathcal{O}^{\mathcal{M}} \not\models A'' \sqsubseteq B$;

where $A, B \in N_C(\mathcal{O}_1)$, $A', A'' \in N_C(\mathcal{O}_2)$.

Concerning the work of Gross *et al.* [GDRH$^+$13], different strategies are considered. Let again be $\mathcal{O}_1$ and $\mathcal{O}_2$ the pair of input ontologies, and let $\mathcal{M}$ be an alignment between them. For sake of conciseness we restrict to the case where only one of the two input ontologies is updated (for instance, we consider $\mathcal{O}_2'$ as the updated version of $\mathcal{O}_2$). The case of two evolving ontologies can be easily reduced to that of a single one with few additional steps (as shown in [GDRH$^+$13]).

Composition-based Approach. This approach is meant for reusing as much as possible the already existing mappings, that are then composed with the new ones. The input of the method

are the three ontologies $\mathcal{O}_1$, $\mathcal{O}_2$ and $\mathcal{O}_2'$, a set of mappings to be deleted $\mathcal{M}_{del} \subseteq \mathcal{M}$ and a set of mappings to be added $\mathcal{M}_{add}$.

The composition takes place between each mappings $m = \langle a, b, r, \_ \rangle \in \mathcal{M}_{del}$ such that a mapping $m' = \langle b, c, r', \_ \rangle \in \mathcal{M}_{add}$ exists, and produces a mapping $m'' = \langle a, c, getNewType(r, r'), \_ \rangle$. The output of getNewType function, for each possible pair of input operands, is given in Table 4.4. We do not discuss here the different strategies for deriving the confidence for the new mapping, because this aspect is not relevant for the analysis at hand.

getNewType	$\equiv$	$\sqsubseteq$	$\sqsupseteq$	$\approx$
$\equiv$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$	$\approx$
$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$\approx$	$\approx$
$\sqsupseteq$	$\sqsupseteq$	$\approx$	$\sqsupseteq$	$\approx$
$\approx$	$\approx$	$\approx$	$\approx$	$\approx$

Table 4.4: Semantic relation composition function getNewType, combining the two operands (from [GDRH+13]).

It is evident that the getNewType function, as well as the composition algorithm, tries to preserve the original semantics, if possible. In any case where the resulting semantic relation is not $\approx$ (*i.e.*, the actual semantics type cannot be automatically determined and should be revised by a domain expert), any conservativity violation tends to be preserved.

Consider the adaptation scenario depicted in Figure 4.6, that shows the preservation of the violation $A \sqsubseteq B$ in ontology $\mathcal{O}_1$. Of course the preservation completely depends on the semantics of the involved mappings. Without loss of generality we restrict our attention to the different configurations of the mappings involving the deleted concept A' only (the different possible configurations for mappings involving concept B' are analogous).

Any configuration involving $\approx$ semantics in one of the two mappings to compose is not listed, because the result is trivially equivalent to $\approx$, which corresponds to mapping removal in the setting considered here (fully automatic adaptation). Any configuration involving $\langle A, A', \sqsupseteq, \_ \rangle$ is ignored as well because, in our considered example, no violation $A \sqsubseteq B$ would exist.

1. $\langle A, A', \sqsubseteq, \_ \rangle$ composed with $\langle A', A'', \equiv, \_ \rangle$ generates $\langle A, A'', \sqsubseteq, \_ \rangle$ (violation is preserved)
2. $\langle A, A', \sqsubseteq, \_ \rangle$ composed with $\langle A', A'', \sqsubseteq, \_ \rangle$ generates $\langle A, A'', \sqsubseteq, \_ \rangle$ (violation is preserved)
3. $\langle A, A', \sqsubseteq, \_ \rangle$ composed with $\langle A', A'', \sqsupseteq, \_ \rangle$ generates $\langle A, A'', \approx, \_ \rangle$ (no adaptation)
4. $\langle A, A', \equiv, \_ \rangle$ composed with $\langle A', A'', \equiv, \_ \rangle$ generates $\langle A, A'', \equiv, \_ \rangle$ (violation is preserved)
5. $\langle A, A', \equiv, \_ \rangle$ composed with $\langle A', A'', \sqsubseteq, \_ \rangle$ generates $\langle A, A'', \sqsubseteq, \_ \rangle$ (violation is preserved)

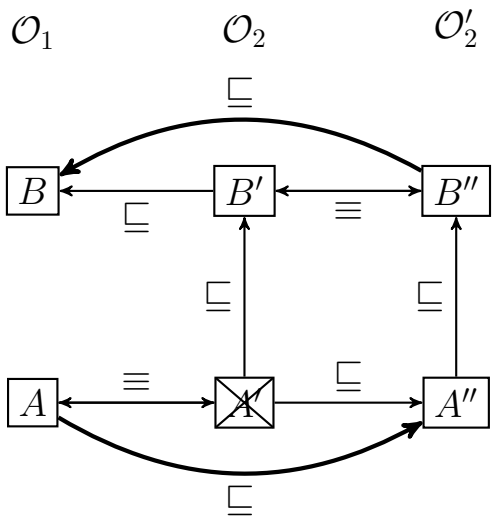


Figure 4.6: Example of composition-based approach adaptation after deletion of concept A'. Bold arcs correspond to adapted mappings.

6. $\langle A, A', \equiv, \_\rangle$ composed with $\langle A', A'', \sqsupseteq, \_\rangle$ generates $\langle A, A'', \sqsupseteq, \_\rangle$ (violation is not preserved)

Note that configuration 5 corresponds to that of Figure 4.6. Only two out of six possible configurations remove the violation, configuration 4 corresponds to the complete mapping removal, while configuration 6 corresponds to mapping weakening (see the repair strategies described in Section 4.7.2). Of course the property of preserving or not a conservativity violation could be used along with other heuristics for the ranking of alternative adaptation strategies. By the analysis of the different possible configurations it is also evident that the proposed adaptation strategy tends to preserve existing violations.

Diff-based Approach. The diff-based approach is centered around the notion of change handler. By analyzing the evolving ontology (*i.e.*, comparing $\mathcal{O}_2$ and its updated version $\mathcal{O}'_2$), the set of occurred change operations is determined (the approach can accomodate any differencing algorithm such as *COnto-Diff* [HGR12] or *PROMPT-Diff* [NM$^+$02]). Each change operation is handled by (possibly more than one) change handler. The authors concretely describe the following set of changes, with associated handlers:

- concept substitution (a concept is renamed),
- concept deletion (a concept is deleted),
- concept merge (different concepts are merged into a single one),

- concept split (a concept is split into several concepts).

The aforementioned handlers are described in Figure 4.9.

Substitution change handler for a change operation renaming an atomic concept A into an atomic concept B (denoted $substitute(A, B)$) transforms each mapping of the form $\langle X, A, r, \_\rangle$ (resp. $\langle A, X', r', \_\rangle$) into $\langle X, B, r, \_\rangle$ (resp. $\langle B, X', r', \_\rangle$). Concept substitution is simply a syntactical renaming without any semantical effect, and it is therefore not interesting for our analysis.

Deletion change handler for a deletion operation of an atomic concept A (denoted as $delC(A)$), is present in two flavours, one deleting each mapping involving A, and the other exploiting the superconcept(s) of A for mapping adaptation. More specifically, for any superconcept A' of A, each mapping of the form $\langle X, A, r, \_\rangle$ is transformed into $\langle X, A', getNewType(r, \sqsubseteq), \_\rangle$. Concept deletion fully corresponds to the adaptation strategy of Martins *et al.*, and all the considerations already formulated apply here as well.

Merge change handler for a merge operation of atomic concepts A, B, C into an atomic concept D (denoted as $merge(\{A, B, C\}, D)$), replaces each mapping of the form $\langle X, Y, r, \_\rangle$, with $X \in \{A, B, C\}$, into $\langle D, Y, getNewType(r, \sqsubseteq), \_\rangle$. Concept merge handler relies on the assumption that D subsumes each of the merged concepts (specifically, it is considered to be equivalent to their union).

Consider the example of Figure 4.7, where change operation $merge(\{A, B, C\}, D)$ has been detected and where a conservativity violation $Y' \sqsubseteq Y$ in $\mathcal{O}_2$ exists, with $A, B, C, D \in N_C(\mathcal{O}_1)$, and $Y, Y' \in N_C(\mathcal{O}_2)$. Arcs between A, B, C and D represents the assumption underlying the algorithm and are shown only for clarity. Mappings $\langle C, Y, \sqsubseteq, \_\rangle$, $\langle A, Y, \sqsubseteq, \_\rangle$ are deleted by the adaptation algorithm. Given that concept D is assumed to subsume each of the merged concepts, axiom $E \sqsubseteq D$ automatically follows from axiom $E \sqsubseteq C$. In addition, the change handler automatically inserts mapping $\langle D, Y, \sqsubseteq, \_\rangle$ because of the deletion of mappings $\langle A, Y, \sqsubseteq, \_\rangle$ and $\langle C, Y, \sqsubseteq, \_\rangle$. Notice that the insertion takes place even if only one of these mappings exists. This insertion, however, results in the preservation of the conservativity violation $Y' \sqsubseteq Y$. If this consequence is unwanted, the application of the merge handler would behave incorrectly.

Split change handler for a split operation of an atomic concept A into atomic concepts B, C, D (denoted as $split(A, \{B, C, D\})$), assumes that concept A subsumes concepts B, C and D. This handler suggests the insertion of a mapping $\langle Y, X, getNewType(r, \sqsubseteq), \_\rangle$, for each mapping of the form $\langle Y, A, r, \_\rangle$, with $X \in \{B, C, D\}$, that is deleted by the algorithm. Two variants for this handler are proposed, the first one suggests one mapping for each value of X, the second one chooses the best candidate (possibly many) based on a concept similarity measure between A and the different values of X.

Consider the example of Figure 4.8, where change operation $split(A, \{B, C, D\})$ has been detected and where a conservativity violation $Y' \sqsubseteq Y$ in $\mathcal{O}_2$ exists, with $A, B, C, D \in N_C(\mathcal{O}_1)$, and $Y, Y' \in N_C(\mathcal{O}_2)$. As in the previous example, the arcs representing the assumption under-

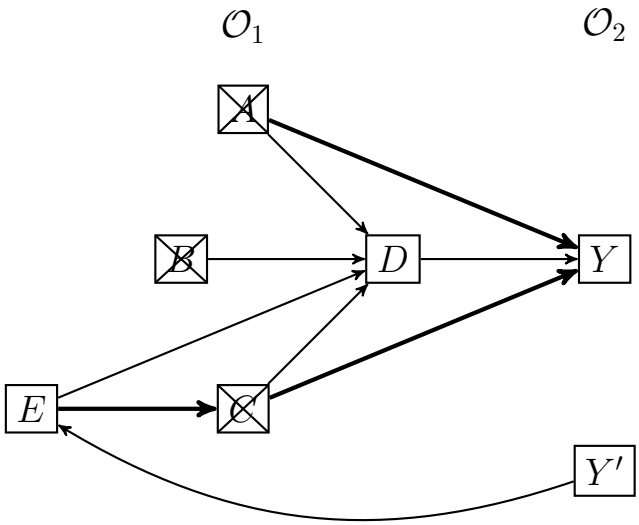


Figure 4.7: Example of diff-based approach adaptation after merge of concepts A, B and C into concept D. Unspecified semantic relations are equal to $\sqsubseteq$.

lying the algorithm (that is, arcs between B, C, D and A) are shown only for clarity. Mapping $\langle A, Y, \sqsubseteq, \_ \rangle$ is deleted by the adaptation algorithm, axiom $E \sqsubseteq A$ is retracted by the modeler, and axioms $E \sqsubseteq C$ and $E \sqsubseteq D$ are inserted. As commonly happens, the fresh concepts (*i.e.*, B, C, D) are integrated by the modeler into the concept hierarchy of the ontology, where they often "inherit" some of the subsumptions involving the target concept (*i.e.*, A). The first variant of the adaptation algorithm will suggest all of the mappings labeled with symbol "?" of Figure 4.8, while the second variant will suggest only the best one w.r.t. the choosen concept similarity metric. By accepting one of $\langle C, Y, \sqsubseteq, \_ \rangle$ and $\langle D, Y, \sqsubseteq, \_ \rangle$, the conservativity violation will hold after mapping adaptation. It is evident that, as shown in this example, an uninformed choice in this context could preserve existing conservativity violations.

Therefore, also for the diff-based adaptation we showed the close relationship and mutual influence between adaptation and (conservativity principle) repair. Despite similar considerations can be analogously formulated for consistency principle, we do not tackle them because this is out of scope of the present thesis.

In order to cope with added concepts, the approach of Gross *et al.* applies an additional matching phase in which new mappings are sought, and the detection of possible consistency principle violations could be added as an additional metric for the selection algorithm.

From the analysis of the main proposals coping with ontology alignment adaptation, the drawbacks of applying purely syntactical evolutionary algorithms to semantic data and metadata clearly emerge.

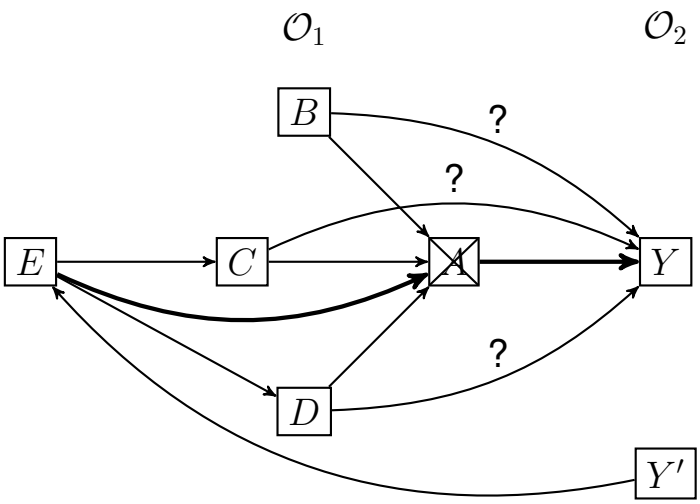


Figure 4.8: Example of diff-based approach adaptation after split of concept A into concepts B, C and D. Unspecified semantic relations are equal to $\sqsubseteq$.

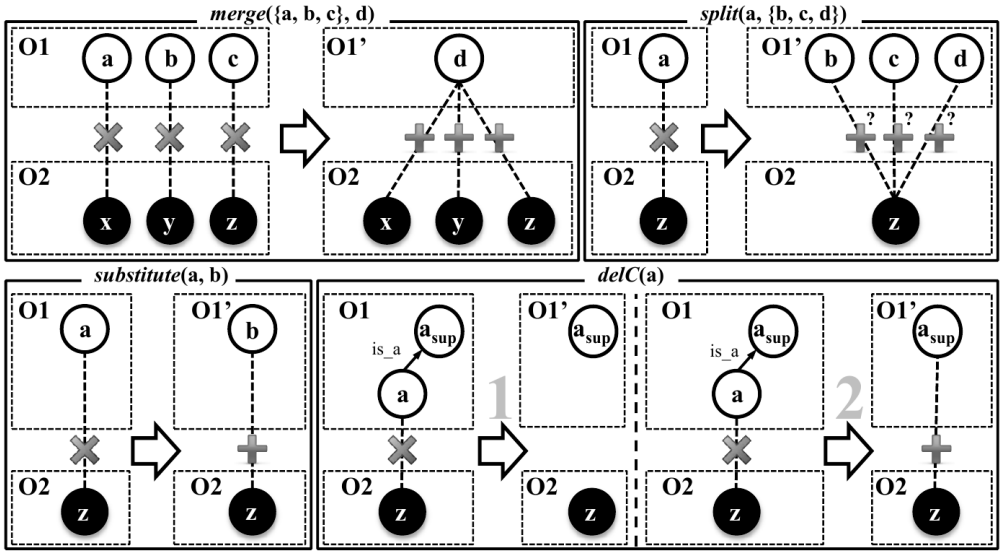


Figure 4.9: Change handlers described in [GDRH+13], picture borrowed from the original article.

Chapter 5

Equivalence Conservativity Principle Violations

5.1 Introduction

As already discussed in Section 4.3.6, the different notions of conservativity principle violation require different solutions. The present chapter is devoted to the subcase of conservativity principle violations w.r.t. equivalence, and it is structured as follows.

First, related work is covered by Section 5.2, Section 5.3 states the problem. The proposed methods and algorithms (including CycleBreaker, the main contribution of the chapter) are presented in Section 5.4, and their experimental evaluation is discussed in Section 5.5. Additional experimental results are provided in [Sol15].

Given that the present chapter is devoted to violations of the conservativity principle w.r.t. equivalence, we generally refer to these violations simply as violations or violations of the conservativity principle. When we refer to other notions we explicitly mention it.

5.2 Related Work

In this section the relevant related work for equivalence conservativity violations is presented. Section 5.2.1 provides an overview of the literature on *Ontology Mapping Debugging*, while Section 5.2.2 presents related work on the *Weighted Feedback Edge Set* problem.

5.2.1 Equivalence Conservativity Principle Violations

Ontology alignment violations (equivalence violations, in particular) are discussed in [BH12], where sanity checks for ontology mappings are proposed, together with general recommendations about best practices in producing ontology mappings (*e.g.*, use of URI for identifying ontologies and matched elements). While checks 6 and 7 are particularly meaningful for our analysis, we do not consider any further the others, because they are not relevant for the problem at hand. Check 6 coincides with pattern (b.ii) of Lily matcher (see Section 4.6.4), that is, the entailment of novel equivalences, among entities of the same input ontology, caused by the presence of multiple mappings sharing a common entity.

This check is analyzed in the context of an exclusive use of "$\equiv$" semantic relation for an alignment between an ontology of mouse anatomy and the corresponding one related to human beings (*NCI* ontology[1]). In none of the 39 detected cases, the target entities were stated equivalent in the input ontology. In a later release of such ontology, 15 entities were merged, while 18 were judged as not equivalent by domain experts (*NCI* ontology curators). This suggests that it is not possible to exclude that (new) correct equivalences may be derived using ontology matching. However, it does not seem the most common case (the evidence of this statement is not only supported by the cited example, but also from the experimental verification described in Section 5.5.8 on *UMLS* alignments). In addition, an indiscriminate use of ontology mappings entailing new equivalences among elements of the same input ontology could be risky. Consider, for instance, a data integration system built on ontology mappings. Suppose that the employed matcher produces an alignment where an equivalence between classes A and B, that are not (originally) equivalent, is entailed. In this context, even if A is correctly matched with class A', the final result would be incorrect, given that all the instances belonging to B would be migrated to A', even if no mapping exists between these two classes. Therefore, in our opinion, although discovering new relations between entities of the same input ontology could be useful in a manual exploratory phase, their indiscriminate use is only advisable for contexts in which correctness is not mandatory (*e.g.*, information retrieval). In general, for the most common usage scenarios of ontology matching (that is, data integration and query answering), a conservative approach in accordance with the conservativity principle seems more appropriate. For what concerns manual check of new equivalences, we aim at equipping our conservativity violation repair tool with a GUI, based on the technique presented in this chapter, for supporting domain experts in the detection (and possible correction) of these novel entailments, by reducing the needed effort and increasing the effectiveness and precision. Check 7, instead, refers to conservativity principle violations w.r.t. subsumption (covered by Chapter 6), for which similar considerations apply.

Another relevant approach is the one proposed by Arnold *et al.* [AR13]. Despite not being directly linked to the conservativity principle and to ontology debugging in general, the proposed algorithm could mitigate the presence of violations by refining the alignment produced by on-

[1] http://www.cancer.gov/cancertopics/cancerlibrary/terminologyresources

tology matchers. The proposed method takes as input the alignment and optional background knowledge and, using a voting algorithm between different linguistic-based heuristics, it aims at refining "$\equiv$" semantic relations to "$\sqsubseteq$"/"$\sqsupseteq$" ones (and never the other way round). The effect of this refinement, in the graph representation of the aligned ontology we propose in this chapter, removes some edges, and this removal could break some problematic cycles. Our algorithm, instead, analyzes the alignments under the point of view of graph connectivity, and it computes a diagnosis for all the problematic SCCs (corresponding to violations of the conservativity principle). For these reasons we believe that the approach by Arnold *et al.* could be employed as an intermediate step between ontology matching and ontology debugging, in order to exploit linguistic knowledge to refine the alignments and, possibly, already reducing the number of problematic cycles in the aligned ontology.

5.2.2 (Weighted) Feedback Edge Set Problem

To the best of our knowledge, no suitable heuristics have been proposed for approximating the *WFES* Problem. An heuristic based on *Simulated Annealing* has been proposed by Galinier *et al.* [GLB13] for solving the *FVS* Problem. Exploiting the equivalence between *FES* and *FVS*, their approach can solve both problems using a reduction, but unfortunately it cannot be trivially extended to the weighted generalization of the problems, due to their local search mechanism. On the contrary, exact and optimal approximations to the problem have been proposed [ENSS98].

5.3 Problem Statement

This section introduces basic notations, definitions and relevant properties of aligned ontology graph representation (Section 5.3.1), a discussion of diagnosis properties (Section 5.3.2) and the formal definition of the problem that aims at computing a minimal diagnosis taking as input an aligned ontology violating the conservativity principle (Section 5.3.3). The section concludes discussing relationships of multiple mappings filtering for aligned ontologies and diagnosis computation w.r.t. conservativity principle (Section 5.3.4).

5.3.1 Notations, Definitions and Properties

In this section we introduce the basic notations and definitions for the graph representation and related concepts, that will be used for mapping repair.

We assume that $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ is the graph representation of an aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$. We omit the superscript and/or the subscript when they are clear from the context. We denote with

$SCC(G(\mathcal{O}^{\mathcal{M}}))$ the set of SCCs of $G(\mathcal{O}^{\mathcal{M}})$, and with $SCC_{local}(G(\mathcal{O}^{\mathcal{M}}))$ the set of SCCs of $G(\mathcal{O}^{\emptyset})$.

Aligned Ontology.

Definition 5.1 formalizes a variant of the aligned ontology, already introduced in Definition 4.19, that is the starting point for our analysis. This variant is needed in order to enable a more efficient detection phase for equivalence violations.

Definition 5.1. Given two (input) ontologies, namely $\mathcal{O}_1$, $\mathcal{O}_2$, and an alignment $\mathcal{M}$ between them, consider an ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ such that $Sig(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}) = Sig(\mathcal{O}_1) \cup Sig(\mathcal{O}_2)$, and $Axioms(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}) = \mathcal{C}l(Axioms(\mathcal{O}_1)) \cup \mathcal{C}l(Axioms(\mathcal{O}_2)) \cup \mathcal{M}$, where $\mathcal{C}l$ represents the closure w.r.t. logical inference. We define $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ the *aligned ontology* w.r.t. $\mathcal{O}_1$, $\mathcal{O}_2$, and $\mathcal{M}$, or simply the aligned ontology, when no confusion arises. △

We are only interested in a subset of the logical closure (possibly infinite, depending on the DL language employed and the axioms composing the ontology), called *classification*. Given an ontology, the *classification* corresponds to the computation, performed using a reasoner, of the full subsumption/subconcept relation (denoted with symbol $\sqsubseteq$) between its named concepts (*i.e.*, elements of N_C). Classification exploits the axioms containing (anonymous) class expressions for deriving consequences (w.r.t. subsumption) between named classes. Classification is therefore the subset of the logical closure of the ontology $\mathcal{O}$ such that each axiom is of the form $A \sqsubseteq B$, where $A, B \in N_C(\mathcal{O})$. This is motivated, in the context of ontology alignment, by the fact that available ontology matching systems compute mappings between named concepts only.

An example of aligned ontology (as defined in Definition 5.1) is shown in Example 5.1.

Example 5.1. Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two input ontologies such that $\mathcal{O}_1 = \{Laptop \sqsubseteq PC, Ultrabook \sqsubseteq Laptop \sqcap LightObject\}$, $\mathcal{O}_2 = \{Ultrabook \sqsubseteq Notebook, Notebook \sqsubseteq Computer\}$, having signatures equal to $\{Laptop, PC, Ultrabook, LightObject\}$ and $\{Ultrabook, Notebook, Computer\}$, respectively. Given that $\mathcal{O}_1 \models Ultrabook \sqsubseteq Laptop \sqcap LightObject$, the classification of this ontology contains axioms $Ultrabook \sqsubseteq Laptop$, and $Ultrabook \sqsubseteq LightObject$, in addition to axiom $Ultrabook \sqsubseteq PC$ and the explicitly asserted axiom $Laptop \sqsubseteq PC$. Classification of $\mathcal{O}_2$ is equal to $\{Ultrabook \sqsubseteq Notebook, Notebook \sqsubseteq Computer, Ultrabook \sqsubseteq Computer\}$. Let $\mathcal{M} = \{\langle Laptop_1, Notebook_2, \sqsupseteq, 0.7\rangle, \langle Ultrabook_1, Ultrabook_2, \equiv, 1\rangle\}$ be an alignment between $\mathcal{O}_1$ and $\mathcal{O}_2$. The aligned ontology w.r.t. $\mathcal{O}_1$, $\mathcal{O}_2$ and $\mathcal{M}$, namely $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$, is therefore equal to $\{Laptop_1 \sqsubseteq PC_1, Ultrabook_1 \sqsubseteq Laptop_1 \sqcap LightObject_1, Ultrabook_1 \sqsubseteq Laptop_1, Ultrabook_1 \sqsubseteq LightObject_1, Ultrabook_1 \sqsubseteq PC_1, Ultrabook_2 \sqsubseteq Notebook_2, Notebook_2 \sqsubseteq Computer_2, Ultrabook_2 \sqsubseteq Computer_2, Notebook_2 \sqsupseteq Laptop_1, Ultrabook_1 \equiv Ultrabook_2\}$, with signature $\{Laptop_1, PC_1, Ultrabook_1, LightObject_1, Ultrabook_2, Notebook_2,$

$Computer_2\}$. Note that subscripts in the aligned ontology are only used to emphasize the "provenance" of the named concepts of the input ontologies, given that for clarity we do not employ here full URIs (that is, namespaces followed by entity names). ◊

Graph Representation.

CycleBreaker algorithm does not directly compute a diagnosis using the aligned ontology, it rather works on a graph representation[2] of this ontology, presented in Definition 5.2.

Definition 5.2. Given an ontology $\mathcal{O}$, its *graph representation*, denoted as $G(\mathcal{O}) = (V, A)$, is a digraph characterized by the following properties:

- each (named) concept $C \in N_C(\mathcal{O})$ has an associated vertex $v_C \in V$,
- each axiom of the form $D \sqsubseteq E \in Axioms(\mathcal{O})$, with D, E distinct elements, has an associated arc $(v_D, v_E, \_)$ in A,
- each axiom of the form $D \sqsupseteq E \in Axioms(\mathcal{O})$, with D, E distinct elements, has an associated arc $(v_E, v_D, \_)$ in A,
- each axiom of the form $D \equiv E \in Axioms(\mathcal{O})$, with D, E distinct elements, has an associated pair of arcs $\{(v_D, v_E, c), (v_E, v_D, c)\}$ in A.

If $\mathcal{O}$ is not an aligned ontology, the weight of the arcs is always equal to 1. If $\mathcal{O}$ is an aligned ontology, the weight is equal to 1 for the axioms of the input ontologies, and it is equal to the confidence of the considered mapping, otherwise. △

When no confusion arises we interchangeably refer to an ontological concept and its associated vertex, and to an ontology and its associated graph representation. Note that subsumption/equivalence axioms between a concept and itself are not allowed in the graph representation, because the set of arcs is defined using an antireflexive relation. This is further motivated by the fact that allowing self-arcs would have no impact on our approach other than leading to a more intricate formalism. Similarly, we will interchangeably refer to axioms/mappings and their corresponding elements in the graph representation.

An example of graph representation is shown in Example 5.2.

Example 5.2. In Figure 5.1 the graph representation associated with the aligned ontology of Example 5.1 is shown. ◊

[2]Despite several proposals for graph formalisms for representing DL ontologies exists in literature (*e.g.*, [MCHS09]), we provided a simplied variant specifically tailored to capture equivalence violations.

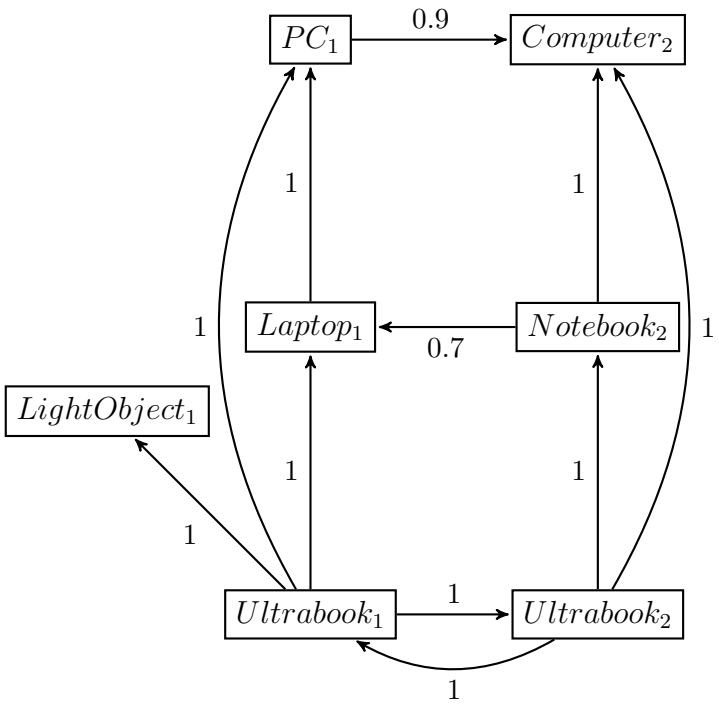


Figure 5.1: Graph representation for the aligned ontology of Example 5.1.

Paths and Cycles.

Given the semantics of the arcs in the graph representation, a path from a vertex u to a vertex v implies that $\mathcal{O}$ entails that the concept associated with u is subsumed by the concept associated with v, as expressed by Proposition 5.1. Analogously, Proposition 5.2 states that a cycle in the graph representation implies that $\mathcal{O}$ entails that all the concepts associated with the vertices of the cycle are equivalent. Note that the other direction of the following propositions holds only if the graph representation is built on top of an ontology closed w.r.t. classification, but this is not the case for our technique. Therefore, we do not require that the graph representation of the aligned ontology reflects all its subsumption entailments (equivalently, we require that using the graph representation for testing subsumption is sound but not necessarily complete).

Proposition 5.1. Let $\mathcal{O}$ be an ontology and let $G(\mathcal{O}) = (V, A)$ be its graph representation. If a path $\pi = [v_{A_1}, \ldots, v_{A_n}]$, with $n > 1$ exists in $G(\mathcal{O})$, then $\mathcal{O} \models A_1 \sqsubseteq A_n$, with A_1, A_n distinct concepts.

Proof. The proof directly follows from the semantics of the arcs in the graph representation and by transitivity of the subsumption relation.

□

An example of correspondence between paths in a graph representation and associated subsumptions in the aligned ontologies is shown in Example 5.3.

Example 5.3. Consider the aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ of Example 5.1, and its graph representation, shown in Example 5.2. We have that $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models Ultrabook_2 \sqsubseteq Computer_2$, given that axiom $Ultrabook_2 \sqsubseteq Computer_2$ appears in the classification of $\mathcal{O}_2$. In this case, the reasoner computing the classification of the ontology $\mathcal{O}_2$ infers this axiom due to transitivity property of subsumption. As an effect of the entailment $Ultrabook_2 \sqsubseteq Computer_2$ by $\mathcal{O}_2$, the corresponding path $[Ultrabook_2, Notebook_2, Computer_2]$ exists in $G(\mathcal{O}_2)$, the graph representation of the ontology $\mathcal{O}_2$, as required by Proposition 5.1. $\Diamond$

Proposition 5.2. Let $\mathcal{O}$ be an ontology and let $G(\mathcal{O}) = (V, A)$ be its graph representation. If a cycle $\kappa = [v_{A_1}, \ldots, v_{A_n}, v_{A_1}]$ with $n > 1$ exists in $G(\mathcal{O})$, then $\mathcal{O} \models A_i \equiv A_j$, with $i, j \in [1 \ldots n]$ and A_i, A_j distinct concepts.

Proof. Given a pair of vertices v_{A_i}, v_{A_j}, we define two paths $\pi_{ij} = [v_{A_i}, \ldots, v_{A_j}]$ and $\pi_{ji} = [v_{A_j}, \ldots, v_{A_i}]$ such that vertices v_{A_p}, v_{A_q} are consecutive in π_{ij}, π_{ji}, only if the same holds in κ, with $i, j, p, q \in [1 \ldots n]$. Given that all the π_{ij} and π_{ji} are paths, we can apply Proposition 5.1.

□

An example of correspondence between equivalence axioms in an aligned ontologies and associated cycles in the graph representation is shown in Example 5.4.

Example 5.4. Consider the graph represented in Figure 5.2. It is a subgraph of the aligned ontology $\mathcal{O}^{\mathcal{UMLS}}$ representing *UMLS* 2012 alignment, namely $\mathcal{UMLS}$, between ontologies $\mathcal{O}_1$, $\mathcal{O}_2$, corresponding to *FMA*[3] and *NCI*[4] ontologies, respectively, representing biomedical knowledge about human anatomy. We have that $\mathcal{O}_2 \not\models Interleukin\text{-}3_2 \equiv Hematopoietic\_Growth\_Factor_2$, because no cycle containing both $Interleukin\text{-}3_2$ and $Hematopoietic\_Growth\_Factor_2$ exists in $G(\mathcal{O}_2)$, and *viceversa* (remind that $\mathcal{O}_2$ is closed by classification). Instead, given that cycle $[Interleukin\text{-}3_2, Interleukin\_3_1, Hematopoietic\_Growth\_Factor_2, Interleukin\text{-}3_2]$ exists in $G(\mathcal{O}^{\mathcal{UMLS}})$, we know that $\mathcal{O}^{\mathcal{UMLS}} \models Interleukin\text{-}3 \equiv Hematopoietic\_Growth\_Factor$. $\Diamond$

In Definition 5.3 safe cycles are introduced, that is, cycles of the aligned ontology that do not violate the conservativity principle. This notion is key to disciminate between cycles representing or not an equivalence violation, and it is also at the basis of the encoding of equivalence violations in terms of our graph representation. Intuitively, a cycle is safe if each projection on one of the two input ontologies is either a single vertex, or a cycle traversing all the vertices of such projection exists. A special case of safe cycles is represented by cycles already existing in the

[3] http://sig.biostr.washington.edu/projects/fm/AboutFM.html

[4] http://www.cancer.gov/cancertopics/cancerlibrary/terminologyresources

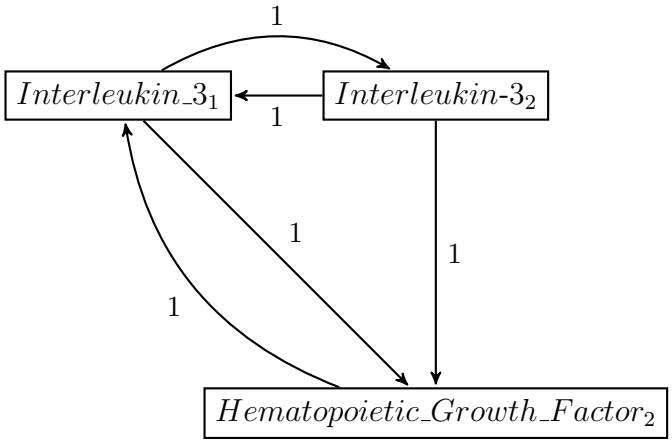


Figure 5.2: Example of cycle defined by *UMLS* 2012 alignment between *FMA* and *NCI* input ontologies. The equivalence of the three vertices is entailed by the aligned ontology.

input ontologies. The different kinds of safe cycles stem from the combination of the following conditions on the two projections of the cycle over the input ontologies: a projection is empty (condition (i)), a projection consists of a single vertex (condition (ii)), a cycle traversing all the vertices of a projection exists (condition (iii)). For instance, a (safe) cycle is local to the input ontology $\mathcal{O}_i$ if its projection on $\mathcal{O}_i$ satisfies condition (iii), while the other projection is empty (condition (i holds). This implies an equivalence entailment (involving all the concepts of the cycle) already in the input ontology $\mathcal{O}_i$, that clearly prevents the existence of an equivalence violation among such concepts.

Definition 5.3. Let $\kappa = [u_1, \ldots, u_n, u_1]$ be one of the cycles of a graph representation $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$, with $n > 1$. We define κ as a *safe cycle* iff *(i)* $|\mathcal{V}(\kappa) \cap V_{\mathcal{O}_i}| = 0$, or *(ii)* $|\mathcal{V}(\kappa) \cap V_{\mathcal{O}_i}| = 1$, or *(iii)* a cycle κ' in $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})|_{V_{\mathcal{O}_i}}$ exists, such that $\mathcal{V}(\kappa) \cap V_{\mathcal{O}_i} \subseteq \mathcal{V}(\kappa')$, for each $i \in \{1, 2\}$. We further differentiate between *local* ((i) and (iii) hold), *trivial* (only (ii) holds), *partially trivial* ((ii) and (iii) hold) and *nontrivial safe cycles* (only (iii) holds). We generically refer to safe cycles, except local safe cycles, as *global safe cycles*. A cycle that is not safe is defined as an *unsafe cycle*. $\triangle$

In Example 5.5 an example is provided for each different kind of cycles defined above.

Example 5.5. Consider the graph representation of Figure 5.3. We first present the (different kind of) safe cycles. $[Food_2, Nourishment_2, Cooking_2, Food_2]$ is a local safe cycle, $[Food_1, Food_2, Nourishment_2, Food_1]$ is partially trivial, $[Food_1, Food_2, Food_1]$ is trivial, and $[Food_2, Nourishment_2, Food_1, Food_2]$ is nontrivial.
Instead, $[Food_2, Nourishment_2, PetFood_1, Food_1, Food_2]$ is an unsafe cycle. $\Diamond$

Starting from the results of Proposition 5.1 and Proposition 5.2, we can characterize a restricted version of the conservativity principle using graph-theoretical concepts only, applied on the graph

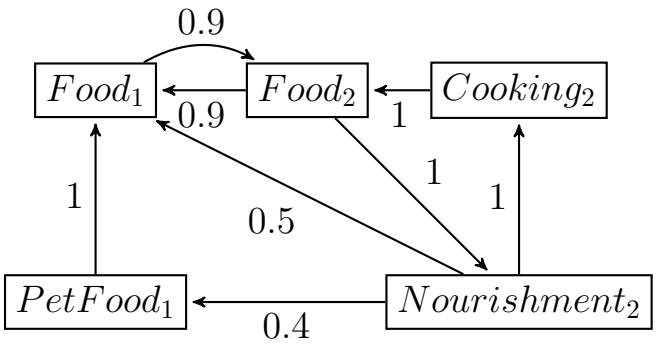


Figure 5.3: A graph representation including both safe and unsafe cycles.

representation, without the need to refer to the aligned ontology. In addition, once the graph representation is computed, we do not need to use reasoners, one of the main sources of intractability. We remark that only two kinds of new consequences can be inferred, subsumption axioms, and equivalence axioms (that is, pairs of subsumption axioms).

The other direction of Proposition 5.1 and Proposition 5.2 (that is, the existence of a path/cycle reflecting any subsumption/equivalence axiom) is only valid for graph representations built from ontologies classified with a full reasoner. In our setting, in practice, they only hold for the input ontologies, given that the aligned ontology is built using the classification of the input ontologies, with the addition of the axioms contributed by the mappings, but it is not classified after this composition. This is not sufficient for capturing more complex violations entailed in the aligned ontology. However, we can check if an equivalence between two named concepts was already existing in the input ontologies, by only using the graph representation. Our conservativity principle detection approach is therefore sound but incomplete, not only because the conservativity principle does not coincide with the convervative extension problem, but also for the lack of classification in the aligned ontology. Note that we do not perform classification of the aligned ontology because in this way we would lose the guarantee of detecting only violations resulting in SCCs which include the mapping responsible for the violation itself. This condition is necessary for applying the diagnosis computation techniques discussed in Section 5.3.3. As a consequence, our conservativity principle detection technique is a sound heuristic but does not provide completeness guarantee. Despite the aforementioned approximations, our repair technique for equivalence violations is effective in practice, as experimentally verified in Section 5.5.3.

Theorem 5.1. Let $\mathcal{O}_1, \mathcal{O}_2$ be two (input) ontologies, and let $\mathcal{M}$ be an alignment between them. Let also $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}) = (V, A)$, $G(\mathcal{O}_1) = (V_{\mathcal{O}_1}, A_{\mathcal{O}_1})$ and $G(\mathcal{O}_2) = (V_{\mathcal{O}_2}, A_{\mathcal{O}_2})$ be the graph representations associated with $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$, $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively. If a path $\pi = [v_1, \ldots, v_n]$ in $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ exists, and no path $\pi' = [v_1, \ldots, v_n]$ (not necessarily different from π) exists in $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})|_{V_{\mathcal{O}_i}}$, for each $i \in \{1, 2\}$ such that $v_1, v_n \in V_{\mathcal{O}_i}$, a violation of the conservativity principle w.r.t. subsumption exists in $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$. If an unsafe cycle κ^u in $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ exists, and

the vertices associated with concepts C, D, namely V_C, V_D, belong to κ^u, then a violation of the conservativity principle w.r.t. equivalence, involving concepts C, D, exists in $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$.

Proof. Proposition 5.1 and Proposition 5.2 relate paths (resp. cycles) in the graph representation to subsumption (resp. equivalence) axioms in the aligned ontology. We have only soundness guarantee, given that without classifying the aligned ontology equivalences that are not reflected in the graph representation may exist. For proving soundness, we need to verify that the paths (resp. cycles) mentioned in the theorem effectively encode violations of the conservativity principle.

Path $\pi = [v_1, \ldots, v_n]$ in $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ corresponds to $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models v_1 \sqsubseteq v_n$, while requiring that no path π' exists such that $\pi' = [v_1, \ldots, v_n] \in G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})|_{V_{\mathcal{O}_i}}$ corresponds to $\mathcal{O}_1 \not\models v_1 \sqsubseteq v_n$ and $\mathcal{O}_2 \not\models v_1 \sqsubseteq v_n$. The conjunction of these three conditions represents a violation of the conservativity principle w.r.t. subsumption.

An (unsafe) cycle κ^u in $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ corresponds to $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models C \equiv D$, for each of the $v_C, v_D \in c$. An unsafe cycle κ^u also requires that no cycle κ' exists in $G(\mathcal{O}_i)$ such that $\mathcal{V}(\kappa^u) \cap V_{\mathcal{O}_i} \subseteq \mathcal{V}(\kappa')$ holds, for any $i \in \{1, 2\}$. This requires that at least one vertex $v_E \in \kappa^u$ exists such that it does not belong to κ'. In turns, this corresponds to $\mathcal{O}_i \not\models E \equiv F$ and $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \models E \equiv F$, for some $E, F \in N_C(\mathcal{O}_i)$, for each $i \in \{1, 2\}$. This represents an equivalence violation, thus proving the soundness of the detected violations using the graph representation.

□

In the remainder of the chapter, when not explicitly stated, we always refer to violations of the conservativity principle in the restricted sense precised in Theorem 5.1.

As a corollary of Theorem 5.1, we have that no cycles of length 2 can, in isolation (*i.e.*, without considering a composed and longer cycle comprising the one of length 2), violate the conservativity principle w.r.t. equivalence, as Corollary 5.3 states.

Corollary 5.3. If a cycle $\kappa^u = [v_1, \ldots, v_1]$ is an unsafe cycle, then its length is at least 3.

Proof. Given that no self-arcs and no self-loops are allowed in the graph representation, no cycles with length less than 2 exist. Assume that cycle $\kappa^u = [v_1, v_2, v_1]$ violates the conservativity principle. By Definition 5.3, cycles that do not traverse any mapping, are local safe cycles (*i.e.*, cycles already present in the input ontologies). It follows that v_1 and v_2 necessarily belong to different input ontologies, and therefore no new equivalence entailment between elements of the same input ontology may occur. This proves the corollary.

□

Theorem 5.1 guarantees that a method detecting and correcting *all* the unsafe cycles on the graph representation is sound w.r.t. violations of the conservativity principle w.r.t. equivalence in the aligned ontology. We remark also that, to this aim, once the graph representation is built, we do not need the support of a standard reasoner. This feature is one of the main ingredients of our approach for reducing the problem intractability.

Diagnoses for Equivalence Conservativity Violations.

The goal of CycleBreaker algorithm is, starting from an aligned ontology, to detect violations to the conservativity principle w.r.t. equivalence and to compute a diagnosis, that is, a subset of the alignment which, once removed, solves all the violations. In order to obey to the well-known *principle of minimal change*, the diagnosis is required to have a minimal weight, w.r.t. a metric capturing the amount of information loss that would follow from the application of the diagnosis. In our context, given that a diagnosis represents the removal of a subset of the mappings, each one having a confidence value, a standard minimality criterion for defining diagnosis weight (the quantity to minimize) is the sum of the weights of the arcs associated with the (removed) mappings. In Definition 5.4 we formalize a diagnosis as the set of arcs of the graph representation of an aligned ontology that, once removed, breaks all the unsafe cycles. As already discussed in Section 4.3.5, only optimal solutions to a problem, in the context of ontology alignment debugging, are usually called diagnoses, otherwise the term repair is employed. Despite we do not have completeness guarantees, we employ the term diagnosis because we refer to the existence of unsafe cycles, that is guaranteed to be an optimal solution.

Definition 5.4. Let $\mathcal{M}$ be an alignment such that $G(\mathcal{O}^{\mathcal{M}})$ has unsafe cycles $\{\kappa_1^u, \ldots, \kappa_n^u\}$. Let also $\Delta \subseteq \mathcal{M}$ be an alignment. Δ is a *diagnosis* for $\mathcal{M}$ iff $\Delta \cap \mathcal{A}(\kappa_i^u) \neq \emptyset$, for each $i \in [1 \ldots n]$, that is, the graph representation $G(\mathcal{O}^{\mathcal{M}\setminus\Delta})$ has no unsafe cycles. $\triangle$

Alternatively, we refer to a diagnosis for $\mathcal{M}$ as the diagnosis for the aligned ontology $\mathcal{O}^{\mathcal{M}}$ or its graph representation $G(\mathcal{O}^{\mathcal{M}})$. Note that, given the monotonicity of (standard) DLs, Definition 5.4 does not need to require that no new unsafe cycles are introduced as a consequence of mapping removal.

In Definition 5.5 and Definition 5.6 we formalize diagnosis weight and diagnosis minimality (required by the principle of minimal change), respectively.

Definition 5.5. Let $\Delta = \{a_0, \ldots, a_n\}$ be a diagnosis for an alignment $\mathcal{M}$. The *weight of a diagnosis* Δ, denoted as $w(\Delta)$, is equal to $\sum_{i=1}^{n} w(a_i)$. $\triangle$

Definition 5.6. Let Δ be a diagnosis for an alignment $\mathcal{M}$. Δ is a *minimal diagnosis* for $\mathcal{M}$ iff there is no diagnosis Δ' for $\mathcal{M}$ such that $w(\Delta') < w(\Delta)$. $\triangle$

Another source of information that could be reduced by diagnosis application is the number of global safe cycles. Note that Theorem 5.1 does not guarantee that *only* unsafe cycles are

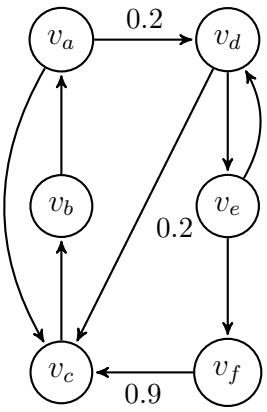


Figure 5.4: Graph representation for which a minimal diagnosis also breaks a global safe cycle. Arcs representing axioms have an omitted weight equal to 1.

broken by the application of a diagnosis. Unfortunately, diagnosis weight minimization could necessarily require to break global safe cycles, as shown in Example 5.6. Clearly, given that local safe cycles do not traverse mappings, by definition, they are not affected by any diagnosis computation strategy.

Example 5.6. Figure 5.4 shows the graph representation of an aligned ontology, with a minimal diagnosis $\Delta = \{(v_a, v_d, 0.2)\}$, with $w(\Delta) = 0.2$. Δ also breaks global safe cycles, such as $[v_a, v_d, v_c, v_b, v_a]$. $\Diamond$

Definition 5.7. Let Δ be a diagnosis for an alignment $\mathcal{M}$. Δ is a *conservative diagnosis* for $\mathcal{M}$ w.r.t. $G(\mathcal{O}^{\mathcal{M}})$ with global safe cycles $\{\kappa_1^s, \ldots, \kappa_m^s\}$ and unsafe cycles $\{\kappa_1^u, \ldots, \kappa_n^u\}$ iff $\Delta \cap \mathcal{A}(\kappa_i^u) \neq \emptyset$, and $\Delta \cap \mathcal{A}(\kappa_j^s) = \emptyset$, for each $i \in [1 \ldots n]$ and $j \in [1 \ldots m]$. That is, all the unsafe cycles of $G(\mathcal{O}^{\mathcal{M}})$ are broken by Δ while all of its global safe cycles are preserved. Any diagnosis that is not conservative is defined as a *nonconservative diagnosis*. $\triangle$

The existence of a nonconservative diagnosis is always guaranteed, as shown in Proposition 5.4, independently from the considered aligned ontology.

Proposition 5.4. Given an aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$, a nonconservative diagnosis Δ always exists, for any pair of ontologies $\mathcal{O}_1$,$\mathcal{O}_2$, and alignment $\mathcal{M}$ between them.

Proof. Nonconservative diagnoses are only required to break all the unsafe cycles of the aligned ontology, and no additional constraints exist. Therefore, $\Delta = \mathcal{M}$ is always a valid diagnosis, thus proving the proposition.

$\square$

Clearly, the additional constraint imposed on *conservative diagnoses* w.r.t. *nonconservative diagnoses* has an impact iff breaking all the unsafe cycles of a graph could (or requires to) break any global safe cycle too. This, in turn, is linked to the possibility, for unsafe cycles, to share some mappings with at least a global safe one.

This aspect is investigated in Proposition 5.5. The proposition also states that the existence property for *conservative diagnoses* is not always guaranteed.

Proposition 5.5. Given an alignment $\mathcal{M}$, the existence of a conservative diagnosis for $\mathcal{M}$ is not guaranteed.

Proof. The minimal requirement for the absence of a diagnosis, is the impossibility to remove all the mappings of at least one unsafe cycle, say κ^u. This situation occurs when all of the mappings of κ^u are shared with global safe cycles. We denote this set of global safe cycles, the *overlapping set of cycles*. The first claim is that there are cases for which no conservative diagnosis exists, independently from the kind of global safe cycles in the overlapping set. In Figure 5.5 we provide different examples of graph representations violating the conservativity principle w.r.t. equivalence, each showing one relevant combination of global safe cycle kinds, for which no conservative diagnosis exists. Details for each single example are given in Example 5.7. From safe cycle definition (Definition 5.3), local safe cycles do not traverse any mapping. This implies that they never share mappings with unsafe cycles. Therefore, computing a diagnosis never requires to break local safe cycles. It follows that, preventing to break local safe cycles does not affect diagnosis existence. This concludes the proof.

□

Example 5.7. In Table 5.1, each relevant combination of global safe cycle kinds for overlapping set is related to its example. In what follows, for each example of Figure 5.5, we detail an unsafe cycle and its overlapping set.

Figure 5.5a: The unsafe cycle is $[A_1, A_2, B_1, A_1]$, with mappings *(i)* $(A_1, A_2, \_)$ and *(ii)* $(A_2, B_1, \_)$. (i) is traversed by trivial safe cycle $[\underline{A_1, A_2}, A_1]$, while (ii) is traversed by trivial safe cycle $[B_1, \underline{A_2, B_1}]$.

Figure 5.5b: The unsafe cycle is $[B_2, A_2, B_1, D_1, B_2]$, with mappings *(i)* $(A_2, B_1, \_)$ and *(ii)* $(D_1, B_2, \_)$. (i) is traversed by nontrivial safe cycle $[\underline{A_2, B_1}, A_1, B_2, A_2]$, while (ii) is traversed by nontrivial safe cycle $[\underline{D_1, B_2}, A_2, E_1, D_1]$.

Figure 5.5c: The unsafe cycle is $[A_1, A_2, B_2, D_1, C_1, B_1, A_1]$, with mappings *(i)* $(A_1, A_2, \_)$ and *(ii)* $(B_2, D_1, \_)$. (i) is traversed by partially trivial safe cycle $[\underline{A_1, A_2}, B_1, A_1]$, while (ii) is traversed by partially trivial safe cycle $[\underline{B_2, D_1}, C_1, B_2]$.

Figure 5.5d: The unsafe cycle is $[A_2, B_2, D_1, A_1, B_1, A_2]$, with mappings *(i)* $(B_2, D_1, \_)$ and *(ii)* $(B_1, A_2, \_)$. (i) is traversed by partially trivial safe cycle $[A_2, \underline{B_2, D_1}, A_1]$, while (ii) is traversed by nontrivial safe cycle $[A_2, B_2, C_1, A_1, \underline{B_1, A_2}]$.

	$\boldsymbol{T}$	$\boldsymbol{PT}$	$\boldsymbol{NT}$
$\boldsymbol{T}$	Figure 5.5a	Figure 5.5e	Figure 5.5e (gray)
$\boldsymbol{PT}$	Figure 5.5e	Figure 5.5c	Figure 5.5d
$\boldsymbol{NT}$	Figure 5.5e (gray)	Figure 5.5d	Figure 5.5b

Table 5.1: Global safe cycle kind combinations for overlapping set w.r.t. an unsafe cycle, and related examples. T means trivial safe, PT partially trivial safe, NT nontrivial safe. (gray) refers to the figure including the gray components.

Figure 5.5e: The unsafe cycle is $[B_1, A_1, A_2, C_1, B_1]$, with mappings *(i)* $(A_1, A_2, \_)$ and *(ii)* $(A_2, C_1, \_)$. (i) is traversed by partially trivial safe cycle $[\underline{A_1, A_2}, B_1, A_1]$, while (ii) is traversed by trivial safe cycle $[\underline{A_2, C_1}, A_2]$.

Figure 5.5e (gray): The unsafe cycle is, as before, $[B_1, A_1, A_2, C_1, B_1]$, with mappings *(i)* $(A_1, A_2, \_)$ and *(ii)* $(A_2, C_1, \_)$. (i) is traversed by nontrivial safe cycle $[\underline{A_1, A_2}, B_1, A_2, B_1, A_1]$, while (ii) is traversed by trivial safe cycle $[\underline{A_2, C_1}, A_2]$. $\Diamond$

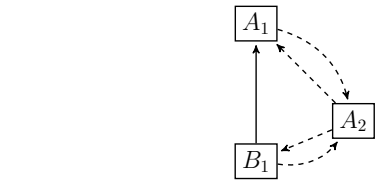
The aim of CycleBreaker algorithm is to compute a minimal diagnosis that is able to remove all the equivalence violations. From Example 5.6 it is evident that the two minimization objectives could be in constrast.

Corollary 5.6. Let Δ be a nonconservative diagnosis, and let Δ' be a conservative diagnosis, for the same alignment $\mathcal{M}$. Then we have that $w(\Delta) \leq w(\Delta')$.

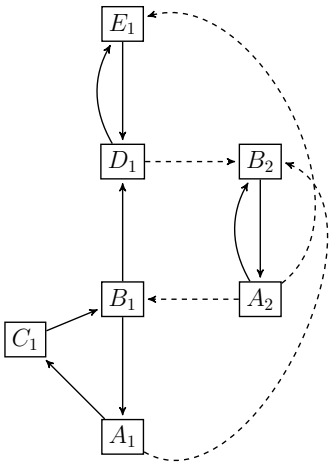
Proof. Let $\{\kappa_1^s, \ldots, \kappa_n^s\}$ and $\{\kappa_1^u, \ldots, \kappa_m^u\}$ be, with $m, n \geq 0$, the set of global safe cycles and unsafe cycles of the considered graph representation, respectively. Assume that $w(\Delta) > w(\Delta')$ holds. Suppose that Δ and Δ' remove the same mappings for breaking unsafe cycles $\kappa_1^u, \ldots, \kappa_{j-1}^u$, with $j \in [1 \ldots m+1]$, written $\Delta_{1,j-1} = \Delta'_{1,j-1}$, with $\Delta_{1,j-1} \subseteq \Delta$, $\Delta'_{1,j-1} \subseteq \Delta'$. It follows that $w(\Delta_{1,j-1}) = w(\Delta'_{1,j-1})$. We denote with $\Delta_{j,m}$, $\Delta'_{j,m}$ the minimal solutions that break the subset of unsafe cycles $\{\kappa_j^u, \ldots, \kappa_m^u\}$ for the two considered variants of the problem, such that $\Delta_{j,m} \subseteq \Delta$, $\Delta'_{j,m} \subseteq \Delta'$. By hypothesis we have that $w(\Delta) > w(\Delta')$, therefore $w(\Delta_{j,m}) > w(\Delta'_{j,m})$ holds.

Clearly, the corollary is applicable when a conservative diagnosis Δ exists. This implies, by definition of conservative diagnosis (Definition 5.7), that it is not necessary to break any global safe cycle. Therefore, for the search problem that aims at computing the conservative diagnosis, the additional constraint leads to a different search space. Therefore, the minimization problem coincides, but the search space is different. In the remainder of the proof, we investigate the relationships between the two search spaces.

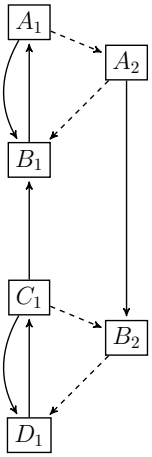
$w(\Delta'_{j,m})$ does not belong to the search space of conservative diagnosis computation problem, otherwise $w(\Delta'_{j,m}) = w(\Delta_{j,m})$ would hold. By assuming that $\Delta_{1,j-1}$ (resp. $\Delta'_{1,j-1}$) does not break unsafe cycles $\{\kappa_j^u, \ldots, \kappa_m^u\}$, $\Delta_{j,m} \cap \Delta_{1,j-1} = \emptyset$ (resp. $\Delta'_{j,m} \cap \Delta'_{1,j-1} = \emptyset$) holds. From



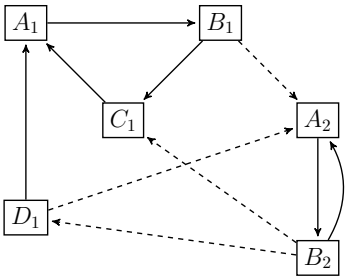
(a) Trivial safe cycles only.



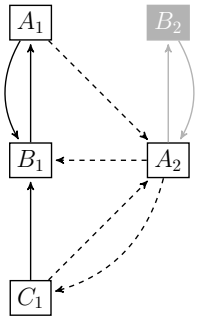
(b) Nontrivial safe cycles only.



(c) Partially trivial safe cycles only.



(d) Partially trivial and nontrivial safe cycles.



(e) Partially trivial (resp. nontrivial, considering light gray additions) and trivial safe cycles.

Figure 5.5: Different examples of problematic graphs without diagnosis in presence of different combinations of global safe cycles, if it is forbidden to break them. Dashed arcs represent mappings.

Definition 5.4 $\Delta_{j,m}, \Delta'_{j,m} \subseteq \mathcal{M}$, and from Definition 5.7 $\Delta'_{j,m} \subseteq \mathcal{M} \setminus (\Delta'_{1,j-1} \cup \bigcup_{k=1}^{n} \mathcal{A}(\kappa_k^s))$. The combination of these two conditions leads to the following constraint on the two search spaces, $\Delta'_{j,m} \subseteq \mathcal{M} \setminus (\Delta'_{1,j-1} \cup \bigcup_{k=1}^{n} \mathcal{A}(\kappa_k^s)) = (\Delta_{1,j-1} \cup \bigcup_{k=1}^{n} \mathcal{A}(\kappa_k^s)) \subseteq \mathcal{M} \setminus \Delta_{1,j-1} \supseteq \Delta_{j,m}$. Given that $\Delta'_{j,m}$ needs to belong to the search space of conservative diagnoses but not to that of nonconservative diagnoses, this leads to a contradiction, because the first is a subset of the second, and this proves the proposition.

□

Therefore, when both minimal conservative and nonconservative diagnoses exist for a given graph, the weight of the latter is less than or equal to the weight of the former. For this reason we introduce, in Definition 5.8, a corresponding minimality definition for conservative diagnoses.

Definition 5.8. Let Δ be a conservative diagnosis for an alignment $\mathcal{M}$. Δ is a *minimal conservative diagnosis* for $\mathcal{M}$ iff there is no diagnosis Δ' for $\mathcal{M}$ such that Δ' is conservative, and $w(\Delta') < w(\Delta)$. △

Example 5.8 shows the corresponding minimal conservative diagnosis, for the same graph of Example 5.6, having a higher weight than the corresponding nonconservative one.

Example 5.8. Consider the graph representation of Figure 5.4. It is easy to see a minimal conservative diagnosis $\Delta' = \{(v_f, v_c, 0.9)\}$ exists, with $w(\Delta') = 0.9$. If we compare this to the weight of the corresponding nonconservative diagnosis (shown in Example 5.6), 0.2, we can observe that the latter is lower, and therefore the additional constraint imposed on conservative diagnoses can increase diagnosis weight. ◊

Problematic Strongly Connected Components.

In Definition 5.9 projection of a SCC of an aligned ontology on one of its input ontologies is introduced. On top of this, in Definition 5.10, we define problematic SCCs, that are used to define an efficient detection method for unsafe cycles.

Definition 5.9. Let $S \in SCC(G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}))$ be a SCC. We define the *projection of a SCC S on an ontology $\mathcal{O}_i$*, denoted as $\Pi_{\mathcal{O}_i}(S)$, as $S \cap V_{\mathcal{O}_i}$, where $G_{\mathcal{O}_i} = (V_{\mathcal{O}_i}, A_{\mathcal{O}_i})$ is the graph representation of ontology $\mathcal{O}_i$, for some $i \in \{1, 2\}$. △

Definition 5.10. A SCC $S \in SCC(G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}))$ is *problematic* iff $\Pi_{\mathcal{O}_1}(S) \notin SCC_{local}(G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}))$ $\vee\ \Pi_{\mathcal{O}_2}(S) \notin SCC_{local}(G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}))$. The set of *problematic SCCs* of $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ is denoted as $pSCC(G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}))$. △

In Example 5.9 an example of a problematic SCC is shown.

Example 5.9. Consider the scenario of Example 5.4 and the subgraph shown in Figure 5.2, that represents a problematic SCC. The projection on the input ontology $\mathcal{O}_2$, indeed, cannot belong to its local SCCs, because no cycle between its two vertices exists. It cannot exist outside this SCC, because otherwise the other traversed vertices would belong to the considered SCC too. $\Diamond$

Lily considers problematic any cycle in the aligned ontology. In addition, they also require that both the input ontologies are acyclic, considering this an unsound situation. From both the logical and modeling points of view, this is not always true, as shown in Example 5.10.

Example 5.10. Consider three axioms of a given input ontology $Food \sqsubseteq Nourishment$, $Nourishment \sqsubseteq Cooking$, $Cooking \sqsubseteq Food$, and their digraph representation: $(Food, Nourishment, 1)$, $(Nourishment, Cooking, 1)$, $(Cooking, Food, 1)$. The vertices associated with this set of arcs is a SCC (and therefore present at least one cycle), but it could be perfectly legal, given that all the terms represented by these concepts have a sense that makes them synonyms.[5] This kind of correspondences are usually detected using *Thesauri*, such as *WordNet*[6]. $\Diamond$

For this reason, we consider a more general case, where cycles are allowed in the aligned ontology as long as the associated SCC is not problematic. Problematic SCCs do not necessarily have a negative impact on the satisfiability of the aligned ontology, but they have the side-effect of asserting equivalences between concepts of the same ontology that were not stated as equivalent in the input ontologies. This is a potentially unwanted consequence, as shown in Example 5.11.

Example 5.11. Consider two input ontologies $\mathcal{O}_1 = \{MolarTooth_1 \sqsubseteq Tooth_1\}$ and $\mathcal{O}_2 = \{Tooth19_2 \sqsubseteq MolarTooth_2\}$ and their obvious associated signatures. Consider also the alignment $\{\langle MolarTooth_1, MolarTooth_2, \equiv, 1\rangle, \langle Tooth_1, Tooth19_2, \equiv, 0.7\rangle\}$. The resulting digraph contains a SCC including all the four vertices, but none of the two projections on the input ontologies are SCCs. As a consequence, all these vertices are equivalent in the aligned ontology. From the domain knowledge, however, we know, for instance, that $Tooth_1 \sqsubseteq MolarTooth_1$ does not hold, as well as $Tooth_2 \sqsubseteq Tooth19_2$, even if they are entailed by the aligned ontology. $\Diamond$

On top of problematic SCCs we define the notion of problematic mappings (Definition 5.11), that is the set of mappings between vertices of a problematic SCC. Intuitively, problematic mappings define the arcs that can appear in diagnoses.

Definition 5.11. Given a graph representation $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$, the set of *problematic mappings* is a subset of the alignment $\mathcal{M}$, denoted as $\mathcal{M}^p$, such that for each $S \in pSCC(G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}))$, and for each $u, v \in S$, if $(u, v, \_) \in \mathcal{M}$, then $(u, v, \_) \in \mathcal{M}^p$. $\triangle$

In Example 5.12 an example of problematic mappings is shown.

[5] See, for instance, `http://www.thefreedictionary.com/food`

[6] English Thesaurus, more information available at `http://wordnet.princeton.edu/`

Example 5.12. Consider the scenario of Example 5.4 and the subgraph shown in Figure 5.2, that represents a problematic SCC. The set of problematic mappings consists in the set of mappings belonging to the problematic SCC, that is: $\{(Interlukin\_3_1, Interleukin\text{-}3_2, 1), (Interleukin\text{-}3_2, Interlukin\_3_1, 1), (Interlukin\_3_1, Hematopoietic\_Growth\_Factor_2, 1), (Hematopoietic\_Growth\_Factor_2, Interlukin\_3_1, 1)\}$. ◊

With the notion of problematic SCCs in place, we can propose an alternative diagnosis definition.

Definition 5.12. Let Δ be a alignment such that $\Delta \subseteq \mathcal{M}$, and let $G(\mathcal{O}^{\mathcal{M}\setminus\Delta}) = (V, A')$ be a graph representation such that $A' = A \setminus \Delta$. Δ is a *diagnosis* for $\mathcal{M}$ w.r.t. $G(\mathcal{O}^{\mathcal{M}})$ iff $pSCC(G(\mathcal{O}^{\mathcal{M}\setminus\Delta})) = \emptyset$. △

From the formulation of diagnosis given in Definition 5.12, we define, in Definition 5.13, the diagnosis for a problematic SCC.

Definition 5.13. Let Δ be a diagnosis for $\mathcal{M}$, let $S \in pSCC(G(\mathcal{O}^{\mathcal{M}}))$ be a problematic SCC, and let Δ' be a nonempty subset of Δ. Δ' is a *diagnosis for a SCC* S iff for each S_i in $SCC(G(\mathcal{O}^{\mathcal{M}\setminus\Delta'}))$ such that $S_i \subseteq S$, we have that $S_i \notin pSCC(G(\mathcal{O}^{\mathcal{M}\setminus\Delta'}))$. △

Clearly, the reformulation does not affect any of the properties of diagnosis we have illustrated during the section. In Example 5.13 a minimal diagnosis for a problematic SCC is shown.

Example 5.13. Consider the scenario of Example 5.4 and the subgraph shown in Figure 5.2, that represents a problematic SCC. A minimal diagnosis for the problematic SCC is $\{(Interleukin\_3_1, Hematopoietic\_Growth\_Factor_2, 1)\}$. ◊

Proposition 5.7 relates unsafe cycles and problematic SCCs, showing that each unsafe cycle results in a problematic SCC.

Proposition 5.7. A SCC is problematic iff it (totally) contains at least one unsafe cycle.

Proof. $\Rightarrow$: Consider a problematic SCC S, with projections Π_1 and Π_2 on the input ontologies. From problematic SCC definition (Definition 5.10), at least one of the projections of S, say Π_1, is not a local SCC. We therefore also know that Π_1 is not a SCC, otherwise it would also be a local SCC. Suppose that Π_1 is a subset of a SCC Π'. This implies that all the elements of Π' belongs to S as well, and therefore Π' and Π_1 are identical, but this contradicts the assumption that Π_1 is not a SCC.

$\Leftarrow$: from the definition of cycle and SCC, each cycle κ is contained in a SCC S (*i.e.*, $\kappa \subseteq S$). Let κ be an unsafe cycle, let also κ_1 (resp. κ_2) be the subset of vertices of κ belonging to an input ontology $\mathcal{O}_1$ (resp. $\mathcal{O}_2$). By definition of unsafe cycle (Definition 5.3), at least one of this subsets, say κ_1, is not contained in any local SCC. But given that $\kappa \subseteq S$, $\kappa_1 \subseteq \Pi_{\mathcal{O}_1}(S)$ holds. Therefore, $\Pi_{\mathcal{O}_1}(S)$ is not contained in any local SCC either and, by Definition 5.10, S is a problematic SCC. This proves the proposition.

□

Proposition 5.7 guarantees completeness for a detection technique for violations of the conservativity principle on a graph representation of an aligned ontology, based on problematic SCCs. Given that all the violations result in unsafe cycles, and that they totally belong to a single problematic SCC, completeness for a repair technique breaking all the unsafe cycles follows.

Notice also that a (unsafe) cycle always belongs to one and only one (problematic) SCC (as expressed by Proposition 5.8), while a problematic SCC may contain more than one cycle. Therefore, a technique detecting problematic SCCs may be more efficient than one directly detecting unsafe cycles in isolation.

Proposition 5.8. A safe cycle cannot traverse more than one SCC of the same input ontology.

Proof. By Definition 5.3, all the vertices belonging to a projection Π of a safe cycle κ^s needs to be traversed by a cycle κ' in the input ontology these vertices belongs to. The claim is that cycle κ' identifies either a SCC of the aligned ontology, or a subset of a SCC. Assume that vertices of Π belong to at least two SCCs S_1, S_2, that is, $\Pi \subseteq S_1 \cup S_2$. Being traversed by a cycle, all the vertices of Π are mutually reachable. Then, from transitivity of reachability, it follows that all the vertices in $S_1 \cup S_2$ are mutually reachable. This contradicts the hypothesis that S_1 and S_2 are two distinct SCCs, thus proving the proposition. Such argument be can straightforwardly generalized to more than two SCCs.

□

5.3.2 Diagnosis Properties

In this section we define desirable properties of diagnoses, and we investigate which ones are satisfied by nonconservative and conservative diagnoses computed using our approach.

1. *Correctness*: a diagnosis needs to be correct, that is, after the application of the diagnosis to the set of data it is computed for, neither the original problems are still present, nor new ones are created by the application of the diagnosis.
2. *Existence*: every alignment has a diagnosis.
3. *Uniqueness*: only one diagnosis exists for an alignment.
4. *Minimality*: this property can be generically expressed as the compliance with the principle of minimal change. In this context, minimality is referred to the diagnosis weight, and the knowledge removed by the diagnosis can be minimized by applying the diagnosis with the lowest weight.

5. *Syntax independency*: the minimal diagnosis coincides for semantically equivalent input ontologies and alignment (that is, for semantically equivalent aligned ontologies).

Nonconservative Diagnoses Properties:

1. *Correctness*: by definition of diagnosis given in Definition 5.4, every diagnosis, and therefore also nonconservative diagnoses, solves all the conservativity principle violations.
2. *Existence*: a nonconservative diagnosis always exists by Proposition 5.4.
3. *Uniqueness*: given an alignment, the minimal diagnosis is not guaranteed to be unique; consider, for instance, Figure 5.2, where $\langle Interleukin\_3_1, Interleukin\text{-}3_2, \sqsubseteq, 1\rangle$, and $\langle Hematopoietic\_Growth\_Factor_2, Interleukin\_3_1, \sqsubseteq, 1\rangle$ are both (minimal) diagnoses.
4. *Minimality*: by definition Definition 5.6, every minimal nonconservative diagnosis minimizes the total sum of mappings weight.
5. *Syntax independency*: if no classification is performed on the input ontologies, and these are not theories (*i.e.*, KBs closed under logical inference), the graph representation is not complete, and not all the unsafe cycles would be detected. Hence, if an incomplete reasoner is employed, its output will not be syntax independent, and so it will be the graph representation and the output of CycleBreaker algorithm.

Conservative Diagnoses Properties: For *conservative diagnoses* the *existence property* may not hold, as shown in Proposition 5.5, as well as the *minimality property*, as defined in Definition 5.8. For the other properties the same considerations used for *nonconservative diagnoses* hold.

In Section 5.3.3 we investigate the computational complexity of the problem at hand.

5.3.3 Problem Complexity

This section introduces the *MAP-WFES* and *CMAP-WFES* problems as extensions of the well-known *WFES* problem which removes unsafe cycles only for solving the equivalence violations, while preserving local cycles. These two problems consist in the computation of a minimal nonconservative and a minimal conservative diagnosis, respectively. The computational complexity of the two problems is then analyzed. To the aim of proving that *MAP-WFES* and *CMAP-WFES* problems are *NP*-hard, we provide polynomial reductions from *WFES* to both of these problems. For clarity of presentation, in this section we sometimes omit weights. From Proposition 5.5 we know that existence is not guaranteed for conservative diagnoses. All the results of this section

are not affected by this property, because they rely on constructions where graphs are free from safe cycles (*i.e.*, any mapping can be removed from the graph), and in such case, *MAP-WFES* and *CMAP-WFES* coincide.

***MAP-WFES* and *CMAP-WFES* Problem Definition.** Let $G = (V, A)$ be a digraph such that $V = V_1 \cup V_2$, with $V_1 \cap V_2 = \emptyset$. We denote $\mathcal{M}$, the set of *mappings*, the subset of A whose arcs are of the form $(u, v, \_)$, such that $u \in V_i, v \in V_j$, for $i, j \in \{1, 2\}$, and $i \neq j$. Let also $\{\kappa_1^s, \ldots, \kappa_n^s\}$ be the set of global safe cycles of G.

MAP-WFES problem aims at computing, given as input the digraph G, a minimum weight feedback edge set $\Delta \subseteq \mathcal{M}$ such that no unsafe cycles exist in $G' = (V, A \setminus \Delta)$.

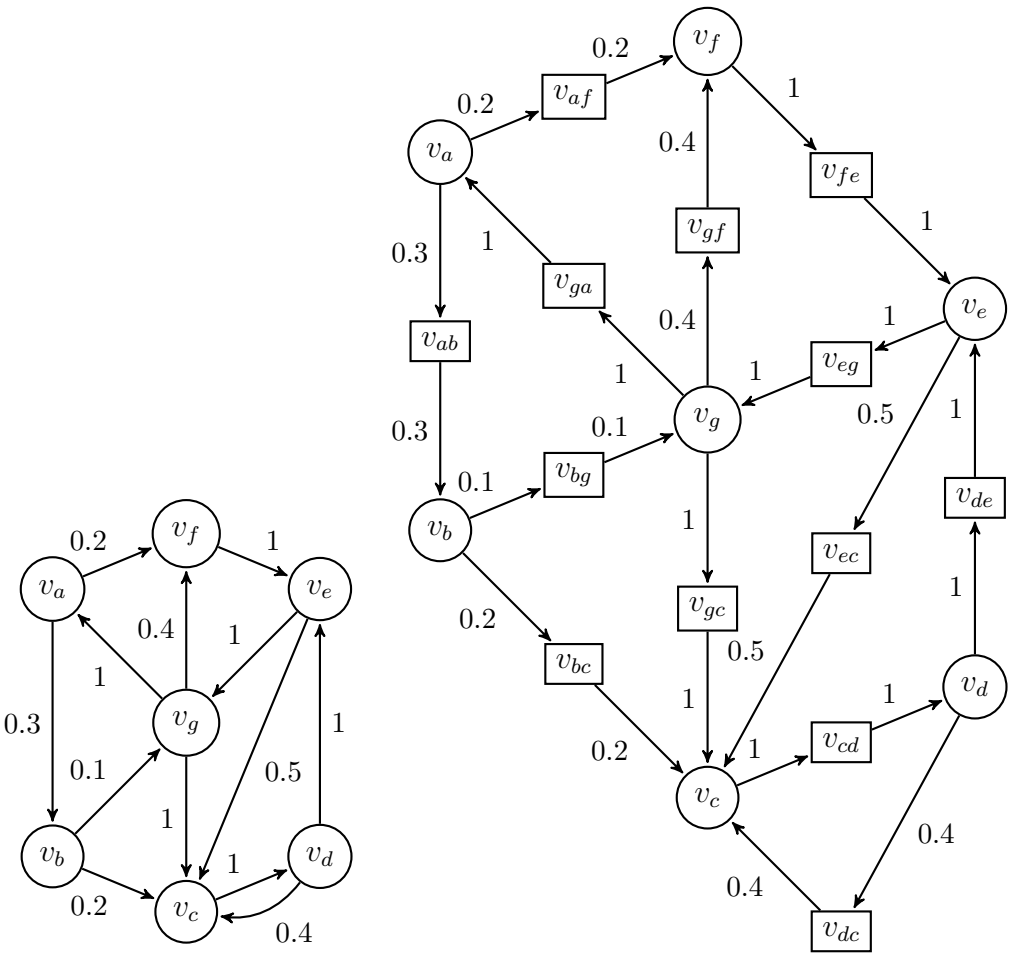
CMAP-WFES problem aims at computing a minimum weight feedback edge set $\Delta \subseteq (\mathcal{M} \setminus \bigcup_{i=1}^{n} \mathcal{A}(\kappa_i^s))$ such that no unsafe cycles exist in $G' = (V, A \setminus \Delta)$.

With the aim of proving that *MAP-WFES* problem is *NP*-hard, Proposition 5.9 introduces a polynomial reduction from *WFES* problem to *MAP-WFES* problem, denoted as *WFES* $\preceq$ *MAP-WFES*. The intuitive idea behind the reduction is the following. Each arc (t, v, c) of the original graph is "split" in two arcs (t, u, c) and (u, v, c), with u a fresh node. All the nodes t, v are associated with one of the input ontology, while the fresh nodes are associated with the other. In this way, all the arcs are mappings (*i.e.*, all of them are potentially removable, exactly as the original arcs). It is easy to see that the reduction preserves the solution weight and that a 1-1 correspondence exists between cycles in the two graphs. In addition, we remark that *MAP-WFES* problem does not break cycles traversing only vertices of one of the input ontologies. No such cycles can exist because all the arcs are mappings, as discussed above. A reduction example is given in Example 5.14, followed by the definition of the reduction in Proposition 5.9.

Example 5.14. In Figure 5.6 graphs G (left) and G' (right) are shown. *WFES* $\preceq$ *MAP-WFES* coincides with G'. The solution to G' is equal to $\Delta = \{(c, cd, 1), (b, bg, 0.1), (gf, f, 0.4), (a, af, 0.2)\}$, with a total weight of 1.7. $D = \{(b, g, 0.1), (g, f, 0.4), (a, f, 0.2), (c, d, 1)\}$ is the corresponding solution to the instance of the *WFES* problem represented by G, and can be easily verified that is both minimal (having weight 1.7) and correct. It is also immediate to see that *WFES* $\preceq$ *CMAP-WFES* coincides with *WFES* $\preceq$ *MAP-WFES*. Therefore, all the above considerations apply to this case as well. $\Diamond$

Proposition 5.9. *WFES* $\preceq$ *MAP-WFES*. A polynomial reduction from the *WFES* problem to the *MAP-WFES* problem exists. Let $G = (V, A)$ be a digraph. The reduction consists in constructing a digraph $G' = (V', A')$ such that a subset of edges, namely $\Delta \subseteq A$, is a solution to *MAP-WFES* problem iff the corresponding set of arcs, namely D, is a solution to the *WFES* problem on G. The reduction is as follows:

1. for each $(x, y, c) \in A$, we create a fresh vertex v_{xy}, we add it to V_2', and we create a pair of arcs $(x, v_{xy}, c), (v_{xy}, y, c)$ that are added to A' and $\mathcal{M}$,



(a) Graph G (reduction input).

(b) Graph G' (reduction output).

Figure 5.6: Input graph G for *WFES* problem (left) reduced to a corresponding graph G', input for the *MAP-WFES*/*CMAP-WFES* problem (right). In graph G', all the round-shaped vertices belong to V_1', while square-shaped vertices belong to V_2'.

2. $V'_1 = V$ and $V' = V'_1 \cup V'_2$.

A set of arcs, namely $\Delta \subseteq A'$, is a solution to the *MAP-WFES* problem on digraph G' iff the corresponding feedback edge set D is a solution to the *WFES* G, where G' is computed from G, and for each arc of the form (x, v_{xy}, c) or (v_{xy}, y, c) in Δ, we have a corresponding arc (x, y, c) in D.

Proof. In order to prove the correctness of the reduction, we need to show that, if $G \preceq G'$, with G an instance of the *WFES* problem and G' an instance of the *MAP-WFES* problem, a set of arcs Δ is a (minimal) solution to G' iff the corresponding set of arcs D is a (minimal) solution to G. As discussed in [ENSS98], the proposed reduction is polynomial and it preserves graph connectivity and the weight of the solutions, by preserving a 1-1 correspondence between cycles of G and those of G', due to the 1-1 correspondency between the arcs of A and those of A'.

$\Rightarrow$: If D is a solution to *WFES* on G, Δ is a solution to *MAP-WFES* on G'. Suppose that Δ is not a solution. This requires that, either at least a cycle κ' exists in digraph $(V', A' \setminus \Delta)$ or that a diagnosis Δ' exists such that $w(\Delta') < w(\Delta)$. For the first case, given the 1-1 correspondence between cycles of G and G', this implies that a corresponding cycle in G exists as well, thus contradicting that D is a solution to the instance of the *WFES* problem represented by G. For the latter case, given the 1-1 correspondence between arcs of G and G', this implies that a solution D' corresponding to Δ' exists. By the weight presevation property of the reduction, $w(D') < w(D)$ holds, contradicting that D is a (minimal) solution to the instance of the *WFES* problem represented by G.

$\Leftarrow$: If Δ is a solution to *MAP-WFES* on G', D is a solution to *WFES* on G. Suppose that D is not. This requires that, either a cycle κ of G exists in digraph $(V, A \setminus D)$, or that a solution D' exists such that $w(D') < w(D)$. The first case requires the existence of a cycle κ' of G' (corresponding to κ) that is left unbroken. In turn, this either violates that Δ is a solution, or that κ' exclusively traverses elements of V'_i, for some $i \in \{1, 2\}$. This situation is excluded by construction of G', because no arcs between vertices of the same subset V'_i of V' exist. For the latter case, this implies that a diagnosis Δ' corresponding to the (minimal) solution D' does not exist. This requires that at least an element of D' cannot belong to a diagnosis (*i.e.*, it cannot be removed). By construction of G', we have that $\mathcal{M} = A'$. Given that only elements of $A' \setminus \mathcal{M}$ cannot belong to a diagnosis for the *MAP-WFES* problem, this results in a contradiction, thus proving the correctness of the reduction.

□

Analogously, Proposition 5.10 introduces a polynomial reduction from *WFES* problem to *CMAP-WFES* problem, for proving that it is an *NP*-hard problem.

Proposition 5.10. *WFES* $\preceq$ *CMAP-WFES*. A polynomial reduction from the *WFES* problem to the *CMAP-WFES* problem exists. This reduction coincides with the one between the *WFES* problem and the *MAP-WFES* problem.

Proof. The only difference between the two problems is that *CMAP-WFES* does not break cycles whose projections on the input subgraphs are subset of local SCCs (*i.e.*, safe cycles). By construction, no SCCs can be present in the reduced graph, given that all the arcs connect elements of the two subgraphs. Therefore, also subgraphs do not contain any SCC. In absence of safe cycles, *CMAP-WFES* problem exactly coincides with *MAP-WFES* problem, therefore the reduction is correct.

□

From the results of Proposition 5.9 and Proposition 5.10 it follows that both *MAP-WFES* and *CMAP-WFES* problems are *NP*-hard, as detailed in Proposition 5.11.

Proposition 5.11. *MAP-WFES* and *CMAP-WFES* are *NP*-hard problems.

Proof. The proof follows from the polynomial reductions to these problems from the *WFES* problem, that is *NP*-hard [ENSS98].

□

Proposition 5.12 relates diagnosis computation to the *MAP-WFES* and *CMAP-WFES* problems introduced in this section. Specifically, it states that computing a nonconservative (resp. conservative) diagnosis for a graph representation reduces to solving *MAP-WFES* (resp. *CMAP-WFES*) problem on it.

Proposition 5.12. Computing a nonconservative (resp. conservative) minimal diagnosis for an aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$, that is, solving all the violations to the conservativity principle w.r.t. equivalence (resp. solving all the violations to the conservativity principle w.r.t. equivalence while not breaking any global safe cycles of $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$), can be reduced to solving an instance of *MAP-WFES* (resp. *CMAP-WFES*) problem.

Proof. From Theorem 5.1 we know that all the violations of the conservativity principle w.r.t. equivalence in the aligned ontology result in unsafe cycles in its graph representation. Therefore breaking all unsafe cycles is equivalent to computing a diagnosis. From the definitions of both problems, we have that no unsafe cycles can exist in the graph resulting from the application of the computed diagnosis (if existing, for *CMAP-WFES* problem). Given that, with the exception of computing a minimal diagnosis, *MAP-WFES* problem has no other constraints, it computes a nonconservative diagnosis. By definition of *CMAP-WFES* problem, instead, the (minimal)

diagnosis search space excludes any mapping belonging to a (global) safe cycle, with the aim of breaking all the unsafe cycles. It is clear that *CMAP-WFES* aims at computing a minimal conservative diagnosis, if it exists. This concludes the proof.

□

Since both *MAP-WFES* and *CMAP-WFES* are *NP*-hard problems (Proposition 5.11), and given the average size of an aligned ontology, computing a diagnosis would be, in practice, intractable. For this reason, our approach decomposes the problem into independent subproblems (*i.e.*, computing a local diagnosis for each problematic SCC), following a *divide et impera* strategy. A (minimal) global diagnosis is then computed from the (minimal) local diagnoses of the single problematic SCCs. Proposition 5.13 proves the correctness of our approach and the optimality of the computed (global) diagnosis.

Proposition 5.13. Computing a (global) diagnosis for a graph G that is nonconservative (resp. conservative), representing an aligned ontology, can be reduced at computing the (local) nonconservative (resp. conservative) diagnoses for the problematic SCCs of G. The (minimal) global diagnosis corresponds to the union of the (minimal) local diagnoses.

Proof. From Proposition 5.7 follows that: *(i)* all and only problematic SCCs contain unsafe cycles, *(ii)* an unsafe cycle does not traverse vertices of more than one SCC (*i.e.*, the unsafe cycles of distinct SCCs are totally disjoint). From (i) completeness follows (it is sufficient to compute a diagnosis for each problematic SCCs to remove all the unsafe cycles in the aligned ontology). (ii) ensures the independence of SCCs, and this guarantees minimality and correctenss for local diagnoses computed in isolation. Finally, it is immediate to see that the minimality property is preserved by the union of local diagnoses, and this concludes the proposition.

□

Proposition 5.13 thus guarantees that the global diagnosis computed as the union of the diagnoses of the problematic SCCs is both minimal and correct (that is, it breaks all the unsafe cycles).

5.3.4 Properties of 1-1 Filtering

This section is devoted to the properties of 1-1 filtering for alignments and their relationships with diagnosis computation. This technique is conceived, in our approach, to derive 1-1 correspondences for problematic SCCs, thus reducing the needed effort for computing a diagnosis, at a higher cost in terms of removed mappings.

The cardinality of the alignment, namely $\mathcal{M}$, influences the size of its associated graph representation. Therefore, tractability also depends on this quantity, and we are interested in determining

an upper bound for the size of $\mathcal{M}$. Proposition 5.14 considers the context in which, given two sets of vertices, no other constraints are imposed on alignment $\mathcal{M}$ between them.

Proposition 5.14. Let U and V be two disjoint sets of vertices such that $|U| = m$ and $|V| = n$. The cardinality of the set $\mathcal{M}$ of distinct mappings between them is the following: $|\{(w, w', \_) \mid (w \in U \wedge w' \in V) \vee (w' \in U \wedge w' \in V)\}| = 2 \cdot m \cdot n$.

Proof. With each of the m vertices of U, we can associate every vertex of V, that are n, so we have $n \cdot m$ distinct pairs $(u, v, \_)$ such that $u \in U, v \in V$. The same number of pairs of the form $(v, u, \_)$ can be identified, thus proving the proposition.

□

Proposition 5.15 shows that 1-1 filtering reduces the upper bound to the size of $\mathcal{M}$.

Proposition 5.15. Let U and V be two disjoint sets of vertices such that $|U| = m$ and $|V| = n$. The cardinality of the set $\mathcal{M}$ of mappings without multiple occurrences of the same vertex in different pairs as the source (resp. target) element is the following: $|\mathcal{M}| = 2 \cdot min(m, n)$.

Proof. The pairs of the form $(u, v, \_)$ and those of the form $(v, u, \_)$, with $u \in U$ and $v \in V$ are independent from each others, so it suffices to prove in isolation that the number of pairs without multiple occurrences of the same element in the source or the target element is equal to $min(m, n)$. We consider here the former case and, without loss of generality, we assume $m \leq n$. With each element u of U we associate a distinct element of V, whose existence is guaranteed by hypothesis, and our constraint prevents the existence of the other $n - 1$ pairs $(u, v', \_)$, with $v' \in V$. In fact, if we subtract from the set of all possible pairs $(u, v, \_)$, with cardinality $m \cdot n$, the pairs with a repetition, that are $m \cdot (n-1)$, we obtain m. Analogously, we have m pairs of the form $(v, u, \_)$, and in total we have $2 \cdot m$ pairs without multiple occurrences in the source or target elements. Given that $m \leq n$ we have that $m = min(m, n)$, thus proving the proposition.

□

The drawback of applying 1-1 filtering, is that the computed diagnosis would also include all the mappings removed during the filtering. The computed diagnosis is therefore usually suboptimal. In addition, 1-1 filtering may also break global safe cycles, as shown in Example 5.15.

Example 5.15. Consider an application of 1-1 filtering on the graph representation of Figure 5.4. In the aforementioned graph, arcs $a_1 = (v_d, v_c, 0.2)$ and $a_2 = (v_f, v_c, 0.9)$, that are mappings, share the target vertex (*i.e.*, v_c). Given that $w(a_1) < w(a_2)$, a_1 is filtered. However, the removal of a_1 breaks partially trivial safe cycle $[v_a, v_d, v_c, v_b, v_a]$, as discussed in Example 5.6. ◊

5.4 Algorithms

This section introduces optimal (Section 5.4.1) and heuristic (Section 5.4.2) methods for computing diagnoses, and a detailed description of CycleBreaker debugging algorithms (Section 5.4.3).

5.4.1 Diagnosis Computation Using *ASP*

In this section we describe how a minimal diagnosis can be computed using *ASP* programs. If not explicitly stated, the diagnoses computed by the presented *ASP* programs are optimal. The concrete syntax we use is compliant with the language accepted by *Lparse 1.0*,[7] a parser for logic programs used as a front-end by different logic programming solvers.

The facts input to the *ASP* problems are of the following kinds:

- Vertices are represented using a binary predicate $vtx(X, O)$, where X is a string representing the vertex label and $O \in \{1, 2\}$ encodes the index of the input ontology the represented concept belongs to,
- arcs are represented using a quaternary predicate $edge(X, Y, C, M)$, where X, Y are vertices (that is, $vtx(X, O)$, $vtx(Y, P)$ hold for some $O, P \in \{1, 2\}$), C is an integer encoding the arc weight in a range $[0 \ldots 100]$, while M is a Boolean flag for differentiating arcs that correspond to axioms ($M = 0$), from that corresponding to mappings ($M = 1$).

When we refer to the execution of an *ASP* program on a graph representation (resp. an aligned ontology), we always implicitly refer to its execution on a set of facts encoding the graph representation (resp. aligned ontology).

Example 5.16. In the following we provide an example of encoding of a graph into a set of facts that can be used in conjunction with the *ASP* programs for solving the *MAP-WFES* and *CMAP-WFES* problems. The encoding of the graph shown in Figure 5.4 corresponds to the set of facts presented in Listing 5.1. The execution of the *ASP* programs of Listing 5.2 and Listing 5.3, in conjunction with the facts of Listing 5.1, produces as output models (always projected on $removed$ predicate, if not specified) $\{removed(edge(a, d, 20, 1))\}$ and $\{removed(edge(f, c, 90, 1))\}$, respectively. These two models correspond, to the minimal diagnoses of Example 5.6 and Example 5.8, respectively. $\Diamond$

Nonconservative Diagnosis Using *ASP*. A minimal nonconservative diagnosis can be computed using the *ASP* program shown in Listing 5.2 (see Proposition 5.16). In what follows we first provide a description of each rule of the *ASP* program:

[7] `http://www.tcs.hut.fi/Software/smodels/`

```
% vertices
vtx(a,1).
vtx(b,1).
vtx(c,1).
vtx(d,2).
vtx(e,2).
vtx(f,2).

% arcs representing axioms
edge(a,c,100,0).
edge(b,a,100,0).
edge(c,b,100,0).
edge(d,e,100,0).
edge(e,d,100,0).
edge(e,f,100,0).

% arcs representing mappings
edge(a,d,20,1).
edge(d,c,20,1).
edge(f,c,90,1).
```

Listing 5.1: *ASP* facts encoding the graph shown in Figure 5.4.

```
% r0
#domain vtx(X,O).
#domain vtx(Y,P).
#domain vtx(Z,Q).

% r1
reaches(X,Y) :- edge(X,Y,C,M), not_removed(edge(X,Y,C,M)), X!=Y.
% r2
reaches(X,Z) :- reaches(X,Y), edge(Y,Z,C,M), not_removed(edge(Y,Z,C,M)), X!=Y, Y!=Z, X!=Z.

% r3
reachesSafe(X,Y) :- edge(X,Y,C,0), O=P, X!=Y.
% r4
reachesSafe(X,Z) :- reachesSafe(X,Y), edge(Y,Z,C,0), O=P, X!=Y, Y!=Z, X!=Z.

% r5
not_removed(edge(X,Y,C,M)) | removed(edge(X,Y,C,M)) :- edge(X,Y,C,M), X!=Y .

% r6
not_removed(edge(X,Y,C,0)) :- edge(X,Y,C,0), X!=Y, O=P.

% r7
unsafeCycle(Y) :- not reachesSafe(Y,X), reaches(Y,X), reaches(X,Y), O=P, X!=Y.

% r8
:- unsafeCycle(Y) .

% r9
#minimize [ removed(edge(X,Y,C,1)) = C ] .

% r10
#hide .
#show removed/1.
```

Listing 5.2: *ASP* program for computing a minimal diagnosis for the *MAP-WFES* problem.

r0: each time variables X, Y and Z appear in the program, they are implicitly associated with the vtx predicate assertion in which they appear,

r1: vertex X $reaches$ vertex Y if an arc $(X, Y, \_)$ exists, and it is not removed,

r2: $reaches$ is a transitive predicate,

r3: vertex X $reachesSafe$ vertex Y iff an arc $(X, Y, \_)$ exists, that is neither a mapping nor removed,

r4: $reachesSafe$ is a transitive predicate,

r5: an arc is either removed or not removed,

r6: removed arcs must be mappings, given that it is not possible to retract axioms of the input ontologies,

r7: a vertex unsafely reaching itself identifies an unsafe cycle,

r8: there must be no unsafe cycles,

r9: soft constraint for diagnosis weight minimization,

r10: the output is restricted to $removed$ predicate only, that is, to the computed diagnosis.

Now we explain how each rule contributes to the encoding of the problem:

r0: the variable domain declaration improves readability and it is used to optimize the grounding phase (it does not directly influence the program semantics, though),

r1,r2: transitive predicate $reaches$ is used to express vertex reachability in the aligned ontology, without further restrictions,

r3,r4: transitive predicate $reachesSafe$ is used to express vertex reachability in the input ontologies, in isolation,

r5: this rule is needed by the *ASP* engine for the model generation,

r6: this rule prunes invalid models removing edges representing axioms of the input ontologies, that are not removable,

r7: this rule identifies unsafe cycles in the graph (see Definition 5.3),

r8: this hard constraint restricts the model space to those breaking all the unsafe cycles (*i.e.*, a diagnosis, by Definition 5.4),

r9: among the possible valid models w.r.t. the constraints imposed by the other rules, this soft constraint chooses one of the models that minimizes the sum of the weights of the removed mappings, thus computing a minimal nonconservative diagnosis (*i.e.*, a minimal diagnosis, see Definition 5.6),

r10: this rule projects the computed model on predicate $removed$, representing the diagnosis, and has no impact on the semantics of the program.

Proposition 5.16 states and proves the equivalence between the minimal nonconservative diagnosis on a digraph, and the solution to the *ASP* program of Listing 5.2 over the facts enconding such digraph.

Proposition 5.16. Given a digraph $G = (V, A)$, its minimal nonconservative diagnosis can be computed using the *ASP* program of Listing 5.2 over the facts encoding G.

Proof. In order to prove the statement, we need to prove the following properties, where u, v are assumed to be elements of V (those same names are used as identifiers in the encoding as *ASP* facts). We denote with G' the digraph equivalent to $(V, A \setminus \Delta)$.

1. $\pi = [u \dots v] \iff reaches(u, v)$, where π is a path in G',
2. for any local safe cycle κ_i^s, $\kappa_i^s \iff reachesSafe(u, v)$ and $reachesSafe(v, u)$ holds, for each $u, v \in \kappa_i^s$,
3. $A = edge(\_, \_, \_, \_)$, $\mathcal{M} = edge(\_, \_, \_, 1)$, $\Delta \subseteq \mathcal{M}$,
4. Δ is minimal,
5. for any unsafe cycle κ_i^u, $\kappa_i^u \iff unsafeCycle(v)$ holds, for any $v \in \kappa_i^u$,
6. $removed(\_) \cap not\_removed(\_) = \emptyset$,
7. $\mathcal{M} = removed(\_) \cup not\_removed(\_)$,
8. $\Delta = removed(\_)$.

1. $\Rightarrow$: we prove this by arithmetic induction on path's length, **base case** for $length(\pi) = 1$, where by the employed encoding, each weighted arc (u, v, c) is translated into a fact $edge(u, v, c, \_)$, and we can apply rule $r1$; **inductive case** for $length(\pi) > 1$, where by inductive hypothesis $\pi_{uv} = [u \dots w] \implies reaches(u, w)$, with $\pi = [u \dots wv]$, and by encoding $(w, v, \_) \implies edge(w, v, c, \_)$ and we can apply rule $r2$;
   $\Leftarrow$: the proof is by arithmetic induction on the number n of rule applications for obtaining the fact $reaches(u, v, \_, \_)$; **base case** for $n = 1$: only $r1$ can be applied and a fact

$edge(u, v, c, \_)$ must exist, by encoding we have that $(u, v, c) \in A \setminus \Delta$; **inductive case** for $n > 1$: by inductive hypothesis we can assume that with $n-1$ rule applications we derived the fact $reached(u, w, \_, \_)$, from which we have $\pi_{uw} = [u \dots w]$, then the last application of rule $r2$ requires the existence of a fact $edge(w, v, c)$ and by the employed encoding this implies the existence of an arc $(w, v, c) \in A \setminus \Delta$, and we obtain $\pi = [u \dots wv]$, as required;

2. this property follows directly from property 1, and by observing that rule $r3$ forces the exclusive "traversal" of edges corresponding to axioms of the input ontologies (*i.e.*, of the form $edge(\_, \_, \_, 0)$);

3. these properties follow trivially from the employed encoding;

4. this property follows from rule $r9$, the semantics of answer sets in presence of optimization constraints, and property 8;

5. $\Rightarrow$: we start by posing the considered unsafe cycle κ_i^u as composed by two set of vertices V_j and V_k, representing its projections over the input ontologies $\mathcal{O}_j$ and $\mathcal{O}_k$, respectively; by definition of unsafe cycle (Definition 5.3), at least one of these two sets is not fully traversed by a (local) cycle, say V_j; this implies that at least two vertices v, v' in V_j exist such that no path $\pi_{vv'} = [v \dots v']$ exists in G; property 2 then implies that *not* $reachesSafe(v, v')$ holds (first atom body); then, from $V_j \subseteq \kappa_i^u$ we have that $reaches(v, v')$ and $reaches(v', v)$ hold (second and third body atoms); therefore, we can apply rule $r7$ deriving facts $unsafeCycle(v)$ and $unsafeCycle(v')$, as required, $\Leftarrow$: from the derivation of the fact $unsafeCycle(v)$ we know that the body of rule $r7$ must be derivable, obtaining facts not $reachesSafe(v, v')$, $reaches(v, v')$, $reaches(v', v)$, from which follows that no path $[v \dots v']$ exists in the digraph $(V, A \setminus \Delta)$ (by property 2), while paths $[v \dots v']$ and $[v' \dots v]$ exist in G (by property 1); this implies, by definition, that a unsafe cycle traversing v and v' exists, as required;

6. this follows from the minimality condition of answer sets (*i.e.*, given that no other rules ever require the membership of an edge to both $removed$ and $not\_removed$ predicates at the same time, any model presenting this situation cannot be an answer set because it cannot be minimal);

7. this follows from the encoding and rule $r5$ which ensures that any edge is either removed or not removed;

8. this directly follows from property 6 and rule $r6$;

9. this follows by the combination of properties 7 and 8.

□

Conservative Diagnosis Using *ASP*. A minimal conservative diagnosis can be computed using the *ASP* program shown in Listing 5.3 (see Proposition 5.17). In what follows we first provide a description of each rule of the *ASP* program:

r0: each time variables X, Y and Z appear in the program, they are implicitly associated with the vtx predicate assertion in which they appear,

r1: vertex X $reachesPre$ vertex Y iff there is an arc $(X, Y, \_)$ in the aligned ontology,

r2: $reachesPre$ is a transitive predicate,

r3: vertex X $reaches$ vertex Y iff an arc $(X, Y, \_)$ exists, and it is not removed,

r4: $reaches$ is a transitive predicate,

r5a-r5b: vertex X $reachesSafe$ vertex Y iff $X = Y$ or an arc $(X, Y, \_)$ exists, and it is neither removed, nor a mapping,

r6: $reachesSafe$ is a transitive predicate,

r7: an arc is either removed or not removed,

r8: removed arcs must be mappings, given that we cannot retract axioms of the input ontologies,

r9: do not remove mappings that are necessary to safe cycles,

r10: a cycle is unsafe iff it creates at least a new cycle involving two vertices belonging to the same input ontology,

r11: there must be no unsafe cycles,

r12: soft constraint for diagnosis weight minimization,

r13: the output is restricted to $removed$ predicate only, that is, to the computed diagnosis.

Now we explain how each rule contributes to the encoding of the problem:

r0: the variable domain declaration improves readability and is used to optimize grounding phase (it does not directly influence the program semantics),

r1,r2: transitive predicate $reachesPre$ is used to express vertex reachability in the aligned ontology considering as empty the set of removed mappings,

r3,r4: transitive predicate $reaches$ is used to express vertex reachability in the aligned ontology, considering only edges, and mappings that are not removed,

```
% r0
#domain vtx(X,O).
#domain vtx(Y,P).
#domain vtx(Z,Q).

% r1
reachesPre(X,Y) :- edge(X,Y,C,M), X!=Y .
% r2
reachesPre(X,Z) :- reachesPre(X,Y), edge(Y,Z,C,M), X!=Y, Y!=Z .

% r3
reaches(X,Y) :- edge(X,Y,C,M), not_removed(edge(X,Y,C,M)), X!=Y .
% r4
reaches(X,Z) :- reaches(X,Y), edge(Y,Z,C,M), not_removed(edge(Y,Z,C,M)), X!=Y, Y!=Z .

% r5a
reachesSafe(X,X) .
% r5b
reachesSafe(X,Y) :- edge(X,Y,C,0), O=P, X!=Y .
% r6
reachesSafe(X,Z) :- reachesSafe(X,Y), edge(Y,Z,C,0), O=P, X!=Y, Y!=Z .

% r7
not_removed(edge(X,Y,C,M)) | removed(edge(X,Y,C,M)) :- edge(X,Y,C,M), X!=Y .

% r8
not_removed(edge(X,Y,C,0)) :- edge(X,Y,C,0), X!=Y, O=P.

% r9
not_removed(edge(A,B,C,1)) :- reachesSafe(X,A), reachesSafe(A,X), reachesSafe(B,Y),
    reachesSafe(Y,B), reachesPre(X,Y), not reaches(X,Y), edge(A,B,C,1), vtx(A,O), vtx(B,P),
    O!=P, X!=Y, A!=B, A!=Y, B!=X .

% r10
unsafeCycle(Y) :- not reachesSafe(Y,X), reaches(Y,X), reaches(X,Y), O=P, X!=Y.

% r11
:- unsafeCycle(Y) .

% r12
#minimize [ removed(edge(X,Y,C,1)) = C ] .

% r13
#hide .
#show removed/1.
```

Listing 5.3: *ASP* program for computing a minimal diagnosis for the *CMAP-WFES* problem.

r5a,r5b,r6: transitive predicate $reachesSafe$ is used to express vertex reachability in the input ontologies in isolation,

r7: this rule is needed by the *ASP* engine for the model generation,

r8: this rule prunes invalid models removing edges representing axioms of the input ontologies, that cannot be retracted,

r9: this rule further restricts the space of the valid models to those not removing mappings needed by global safe cycles (see Definition 5.3), that is, to models that corresponds to conservative diagnoses (see Definition 5.7); in order to be able to correctly identify not only nontrivial safe cycles, but also partially trivial and trivial safe ones, rule $r5a$ is crucial; rule $r5a$, indeed, allows to apply rule $r9$ where $A = X$ holds, as required by partially trivial and trivial safe cycles,

r10: this rule identifies unsafe cycles in the graph (see Definition 5.3),

r11: this hard constraint restricts the model space to those breaking all the unsafe cycles (in combination with rule $r9$ it corresponds to the definition of conservative diagnosis, as given in Definition 5.7),

r12: among the possible valid models w.r.t. the constraints imposed by the other rules, this soft constraint chooses the one (possibly not unique) that minimizes the sum of the weights of the removed mappings, thus computing a minimal diagnosis (see Definition 5.8),

r13: this rule projects the computed model on predicate $removed$, representing the diagnosis, and has no impact on the semantics of the program.

Proposition 5.17 states and proves the equivalence between the conservative diagnosis on a digraph, and the solution to the *ASP* program of Listing 5.3 over the facts enconding such digraph.

Proposition 5.17. Given a digraph G, its minimal nonconservative diagnosis can be computed using the *ASP* program of Listing 5.3 over the facts encoding G.

Proof. Given that the difference between conservative and nonconservative diagnoses is an additional constraint for the former, and that the two corresponding *ASP* programs differ only on this additional constraint, for what concerns the identical subset of the rules we rely on Proposition 5.16. Therefore, in order to prove the statement, we only need to show that the addition of rules $r1$, $r2$, $r5a$ and $r9$ allows to compute answer sets that are exactly the encoding of a conservative diagnosis over the *ASP* encoding of the considered digraph G. The part of the proof relating to rules $r1$ and $r2$ is analogous to that of property 1 of Proposition 5.16, and is therefore omitted. We now need to show how the addition of the aforementioned rules enforces the constraint of not breaking any safe cycle κ_i^s of G. By Definition 5.3, a nonlocal cycle is safe if and only if none of

its two projections (*i.e.*, set of vertices) over the input ontologies, say V_j and V_k, is fully traversed by a local cycle (*i.e.*, a cycle already existing in the considered input ontology), if composed by more than one vertex. A safe cycle is broken when at least one of its arcs is removed. We remind that from properties 8 and 6 of Proposition 5.16 we know that diagnoses correspond to the set of edges belonging to predicate $removed$, and that predicates $removed$ and $not\_removed$ do not overlap, respectively. We only need to show that the two following properties hold.

$removed(edge(u, v, \_, 1)) \implies (u, v, \_) \notin \mathcal{A}(\kappa_i^s)$: suppose that $removed(edge(u, v, \_, 1))$ is derived but $(u, v, \_) \in \mathcal{A}(\kappa_i^s)$. Being κ_i^s a safe cycle, any pair of vertices in its projections, say $\langle x, u\rangle$ and $\langle y, v\rangle$, they are reachable in the input ontologies in isolation, and they therefore belong to predicate $reachesSafe$ (satisfying in these way the first four atoms of rule $r9$). We know that a mapping between u and v exists, and therefore also atom seven of the body is matched. At this point, the necessity of preserving the mapping in order to avoid breaking the safe cycle (*i.e.*, in case of its removal x and y would be reachable in G but not in G'), is translated into facts that are matching body atoms five and six, allowing the application of rule $r9$, and contradicting the hypothesis that $removed(edge(u, v, \_, 1))$ holds.

$(u, v, \_) \in \mathcal{A}(\kappa_i^s) \implies not\_removed(edge(u, v, \_, \_))$: the statement is trivially true if the arc is not a mapping, we therefore exclude this hypothesis. Assume that $(u, v, \_) \in \mathcal{A}(\kappa_i^s)$ but $removed(edge(u, v, \_, \_))$. Being κ_i^s a nonlocal safe cycle, we know that it traverses at least one mapping, say $(u, v, \_)$. For this reason, the seventh atom of the body of rule $r9$ is matched. Being κ_i^s a safe cycle, for any pair of vertices of its projections, say $\langle x, u\rangle$ and $\langle y, v\rangle$, we have that their composing elements are reachable in the input ontologies in isolation, and therefore they belong to predicate $reachesSafe$ (satisfying in these way the first four atoms of rule $r9$). The case in which one of the two projections is composed by a single vertex is handled in conjunction with rule $r5a$, that states that each vertex is safely reachable by itself. Due to the presence of mapping $(u, v, \_)$, and that u is reachable from x, and that y is reachable from v, also atom $reachesPre(x, y)$ is matched. Finally, given that mapping $(u, v, \_)$ is necessary for reaching vertex y from vertex x, also the sixth atom of the body of rule $r9$ is matched, allowing to derive the fact $not\_removed(edge(u, v, \_, \_))$, causing a contradiction.

□

As discussed in Section 5.3, given a graph, a conservative diagnosis may not exist. In such a case, the associated *ASP* program (composed by the facts encoding the graph, and the *ASP* program of Listing 5.3) is unsatisfiable. In order to cope with such cases, we propose a variant of the *ASP* program of Listing 5.3 that replaces the hard constraint of not removing any mapping traversed by a safe cycle into a soft constraint aiming at minimizing their removal (see Listing 5.4).

For sake of conciseness, we only discuss the modified rules, referring to the description of the program of Listing 5.3 for the others.

```
% r0
#domain vtx(X,O).
#domain vtx(Y,P).
#domain vtx(Z,Q).

% r1
reachesPre(X,Y) :- edge(X,Y,C,M), X!=Y .
% r2
reachesPre(X,Z) :- reachesPre(X,Y), edge(Y,Z,C,M), X!=Y, Y!=Z .

% r3
reaches(X,Y) :- edge(X,Y,C,M), not_removed(edge(X,Y,C,M)), X!=Y .
% r4
reaches(X,Z) :- reaches(X,Y), edge(Y,Z,C,M), not_removed(edge(Y,Z,C,M)), X!=Y, Y!=Z .

% r5a
reachesSafe(X,X) .
% r5b
reachesSafe(X,Y) :- edge(X,Y,C,0), O=P, X!=Y .
% r6
reachesSafe(X,Z) :- reachesSafe(X,Y), edge(Y,Z,C,0), O=P, X!=Y, Y!=Z .

% r7
not_removed(edge(X,Y,C,M)) | removed(edge(X,Y,C,M)) :- edge(X,Y,C,M), X!=Y .

% r8
not_removed(edge(X,Y,C,0)) :- edge(X,Y,C,0), X!=Y, O=P .

% r9a
safeR(edge(A,B,C,1)) :- reachesSafe(X,A), reachesSafe(A,X), reachesSafe(B,Y),
    reachesSafe(Y,B), reachesPre(X,Y), not reaches(X,Y), edge(A,B,C,1), vtx(A,O), vtx(B,P),
    O!=P, X!=Y, A!=B, A!=Y, B!=X .
% r9b
#minimize [safeR(edge(X,Y,C,1)) = C @ 1 : edge(X,Y,C,1) : X!=Y : O!=P] .

% r10
unsafeCycle(Y) :- not reachesSafe(Y,X), reaches(Y,X), reaches(X,Y), O=P, X!=Y .

% r11
:- unsafeCycle(Y) .

% r12
#minimize [removed(edge(X,Y,C,1)) = C @ 2] .

% r13
#hide .
#show removed/1.
```

Listing 5.4: *ASP* program for computing an approximation of a minimal diagnosis for the *CMAP-WFES* problem based on removed mappings used by safe cycles.

Rule $r9a$ marks as safely removed (using the unary predicate for arcs $safeR$) the removed mappings that are involved in a safe cycle. Rule $r9b$ aims at minimizing the total weight of such removed mappings. It is only an estimation of the loss of knowledge, not directly related to the number of broken safe cycles, given that they are not directly evaluated.

Other variants may be conceived, for instance by counting the number of concepts that were equivalent for the effect of a safe cycle, and that are no more equivalent when the diagnosis is applied. This can be achieved by introducing a binary predicate $notEQ(X, Y)$, for vertices X and Y, and by replacing the $Head$ of rule $r9a$ with $notEQ(X, Y)$, and by replacing rule $r9b$ with `#minimize[notEQ(X,Y) = 1 @ 1 : X!=Y : O!=P] .`

The complete *ASP* program is detailed in Section C.1, Appendix C, Listing C.1.

Example 5.17 illustrates the difference among the *ASP* program variants proposed in this section.

Example 5.17. Consider the graph representation given in Figure 5.5d, and assume a weight equal to 0.15 for each mapping. On the associated set of facts, the result for each *ASP* program is as follows:

- Listing 5.2: $removed(edge(b2, d1, 15, 1))$,
- Listing 5.3: ***unsatisfiable*** (***i.e.***, no conservative diagnosis exists, as shown in Example 5.7),
- Listing 5.4: $removed(edge(b2, d1, 15, 1)), safeR(edge(b2, d1, 15, 1))$,
- Listing C.1: $removed(edge(b2, d1, 15, 1)), notEQ(b2, d1), notEQ(a2, d1)$. ◊

Example 5.18, instead, shows the difference between nonconservative diagnosis and conservative diagnosis (optimal and approximated) computation.

Example 5.18. Consider the graph representation of Figure 5.5b, and assume a weight equal to 0.3 for each mapping. On the associated set of facts, the result for each *ASP* program is as follows:

- Listing 5.2: $removed(edge(a2, b1, 30, 1))$,
- Listing 5.3: ***unsatisfiable***,
- Listing 5.4: $removed(edge(a2, b1, 30, 1)), safeR(edge(a2, b1, 30, 1))$,
- Listing C.1: $removed(edge(d1, b2, 30, 1)), notEQ(e1, b2), notEQ(e1, a2), notEQ(d1, b2), notEQ(d1, a2)$.

In particular, the minimal nonconservative diagnosis (*i.e.*, $(A_2, B_1, 0.3)$) corresponds to the optimal conservative diagnosis approximation based on the number of removed safe cycles, while it is associated to the suboptimal model $removed(edge(a2, b1, 30, 1))$, $notEQ(b2, c1)$, $notEQ(b2, b1)$, $notEQ(b2, a1)$, $notEQ(a2, c1)$, $notEQ(a2, b1)$, $notEQ(a2, a1)$ for the conservative diagnosis approximation based on the number of lost equivalences. $\Diamond$

5.4.2 CycleBreaker Heuristics

In this section we present two heuristics for computing an approximated solution to an instance of the *MAP-WFES* problem.

Greedy Heuristics. The main strategy followed by the proposed heuristic is to greedily attempt to maximize the number of broken cycles, preferring the mappings with lower confidence among those breaking the same number of cycles. The greedy heuristic, shown in Algorithm 10, takes as input the set of cycles that need to be broken, and an alignment containing all the mappings that may appear in the considered cycles. The set of cycles is computed by the *Johnson's* algorithm [Joh75] (introduced in Section 4.5), from which local safe cycles are removed (by testing if they traverse at least one mapping). The heuristic then returns as output a nonconservative diagnosis breaking all the cycles taken as input.

The relevant steps for the diagnosis computation are described in what follows. To this aim an auxiliary map, named $intercept$, is used to keep track of which cycles a mapping belongs to, that is, the cycles that would be broken by its removal (lines 4–14). An ordering of the mappings in $intercept$ map is computed on the number of associated cycles first (descending), and on the mapping confidence (ascending) then (line 15). The computed ordering it then used for iterating over the mappings in the $intercept$ map (lines 16–23). The output diagnosis is computed by adding mappings (lines 17–19) until no cycles are left unbroken by the diagnosis (line 21).

The condition for adding a mapping to such diagnosis is that the set of cycles associated with the mapping (w.r.t. $intercept$ map) must not be a subset of the set of cycles already broken by the diagnosis at the considered iteration. Intuitively, this condition determines which mappings are useful for the diagnosis (*i.e.*, mappings that break at least a cycle that would be left unbroken by the diagnosis computed so far). The underlying assumption for this heuristic is that, in our setting, it is more common that the optimal (nonconservative) diagnosis w.r.t. diagnosis weight corresponds to the one composed by the least number of mappings.

We remark that this heuristic computes an approximated solution to the *WFES* problem when all the cycles and arcs of a graph are given as input. A similar heuristic for the *FES* problem can be obtained by suppressing the sorting condition on the weight of mappings.

A meta-heuristic can be obtained by leaving unspecified the ordering conditions of line 15. Con-

Algorithm 10 GreedyDiagnosis Function

```
1:  function greedyDiagnosis(Set⟨Cycle⟩ cycles, Alignment M) : Diagnosis
2:      Δ ← ∅
3:      intercept : Mapping × Set⟨Cycle⟩ ← ∅                ▷ Map from mapping to cycles it belongs to
4:      for each κ in cycles do
5:          for v_i, v_{i+1} ∈ κ, with i ∈ [0 ... length(κ)) do
6:              if (v_i, v_{i+1}, _) ∈ M then
7:                  if (v_i, v_{i+1}, _) ∈ intercept then
8:                      intercept((v_i, v_{i+1}, _)) ← intercept((v_i, v_{i+1}, _)) ∪ {κ}
9:                  else
10:                     intercept((v_i, v_{i+1}, _)) ← {κ}
11:                 end if
12:             end if
13:         end for
14:     end for
15:     intercept.sort()          ▷ sort mappings on number of cycles (descending), confidence (ascending)
16:     for each m in intercept do                          ▷ iterates on mappings
17:         if intercept(m) ⊈ ⋃_{m'∈Δ} intercept(m') then
18:             Δ ← Δ ∪ {m}
19:         end if
20:         if cycles ⊆ ⋃_{m'∈Δ} intercept(m') then
21:             break                                       ▷ Diagnosis completed
22:         end if
23:     end for
24:     return Δ
25: end function
```

sider, for instance, a scenario in which it is on average more convenient to greedily minimize the weight. A heuristic for this scenario can be obtained through an instantiation of the meta-heuristic with an inversion of the ordering conditions w.r.t. the heuristic shown in Algorithm 10.

The worst case scenario for the greedy heuristic is represented by any instantiation of the graph pattern shown in Figure 5.7, having $n > 0$. In any of such cases, the diagnosis computed by the greedy heuristic is of the form $\Delta = \{(v_{b_0}, v_{a_0}, c_0)\}$, while a minimal diagnosis of the form $\Delta_{min} = \{(v_{b_k}, v_{a_k}, c_k) \mid k \in [1 \dots n]\}$ exists. Analogously, the same graph pattern represents the worst-case scenario for the greedy heuristics aiming at minimizing the weight: it suffices that $\forall i \in [1 \dots n]$ and $\sum_{i=1}^{n} c_i > c_0$ hold, with $n > 0$, as before. In these situations the heuristic would prefer the suboptimal diagnosis of the form $\{(v_{b_k}, v_{a_k}, c_k) \mid k \in [1 \dots n]\}$, instead of the optimal one, corresponding to the singleton of arc (v_{b_0}, v_{a_0}, c_0).

Genetic Algorithm. Genetic Algorithm [GH88] (GA) is a biologically inspired search meta-heuristic that mimics natural selection. Genetic algorithms belong to the larger class of evolutionary algorithms, which generate solutions to optimization and search problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover of individuals of an evolving population. The main ingredients that characterize a GA are: *(i)* encoding of individuals, *(ii)* cost of individuals, *(iii)* population strategy, *(iv)* mutation stategy for individuals, *(v)* selection stategy for individuals, *(vi)* offspring generation stategy, *(vii)* termination stategies.

We now analyze in detail our GA-based heuristic for approximating the computation of a non-

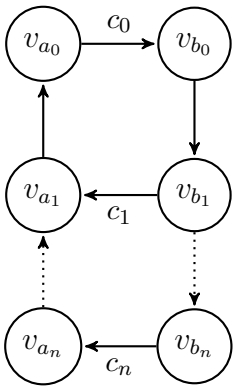


Figure 5.7: With $\sum_{i=1}^{n} c_i < c_0$. The dotted lines represent arcs (v_{b_j}, v_{a_j}, c_j), with $j \in [2 \dots n]$.

conservative diagnosis (that is, the solution to the *MAP-WFES* problem) for a (problematic) SCC.

As a preliminary step, the heuristic requires the enumeration of the simple cycles of the considered SCC. To this aim, *Johnson's* algorithm is employed (local safe cycles are again excluded). The other relevant building blocks of the GA-based heuristic are as follows:

(i) Consider an input SCC S belonging to a graph representation associated with alignment $\mathcal{M}$, the encoding of individuals (resp. simple cycles) is as follows:

1. suppose that S contains m mappings, that is, $|\mathcal{A}(S) \cap \mathcal{M}| = m$, an arbitrary ordering over the mappings of S (resp. c) is defined through a map $r : \mathcal{A}(S) \cap \mathcal{M} \times [1 \dots m]$;
2. the encoding of an individual ind, denoted as $\langle ind \rangle$, is represented through a (Boolean) binary string of length m. The binary strings have character 0 (resp. 1) in the *i-th* position, denoted as $\langle ind|_i \rangle$ iff the *i-th* mapping of S, w.r.t. map r, is excluded from (resp. included in) the corresponding solution.

The same encoding is also used for simple cycles of a SCC. Each individual is required to be a solution, that is, it needs to break all the unsafe cycles of a SCC. More formally, given as input a SCC S and its set of cycles $\{\kappa_1, \dots, \kappa_n\}$, with $n > 0$, for any individual ind the following invariant holds: $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \langle \kappa_i|_j \rangle \wedge \langle ind|_j \rangle = 1$. The invariant implies that each cycle of S is broken, and therefore also the unsafe ones, and the individual represents a correct solution (*i.e.*, a correct nonconservative diagnosis). In the remainder of the section we interchangeably refer to an individual (resp. cycle) and its encoding.

(ii). Given an individual ind, its cost is equal to that of the corresponding diagnosis, that is the sum of the weight of its mappings.

(iii). The population, that is, the individuals considered at each generation, has a fixed size p.

Algorithm 11 fix Function

```
1: function fix(BitString ⟨ind⟩, Set⟨BitString⟩ cyclesEnc) : BitString
2:     for each ⟨κ⟩ in cyclesEnc do
3:         if ⋁_{j=1}^{m} ⟨κ|_j⟩ ∧ ⟨ind|_j⟩ = 0 then
4:             idx ← i ∈ [1 . . . m] s.t.⟨κ|_i⟩ = 1          ▷ Position i is randomly choosen.
5:             ⟨ind|_idx⟩ ← 1
6:         end if
7:     end for
8:     return ⟨ind⟩
9: end function
```

(iv). The mutation strategy in our approach consists in a single mutation probability μ. Given an encoding for an individual, μ represents the complementation probability for each position in the encoding.

(v). Selection strategy is implemented here by a selection of a fixed number r of individuals with the lowest cost, with $r < p$.

(vi). At generation k, $p-r$ individuals, called offspring of generation k, are generated from the set of selected individuals of generation $k-1$. For each element ind of the offspring, two parents ind_1, ind_2 are selected, and a cut-point $cp \in [1 \dots m]$ is randomly chosen, such that for each $i \in [1 \dots cp]$, $\langle ind|_i \rangle = \langle ind_1|_i \rangle$ holds, and for each $j \in [cp+1 \dots m]$, $\langle ind|_j \rangle = \langle ind_2|_j \rangle$ holds. After the generation of each element of the offspring, each position of its encoding is mutated with probability μ. A special case is represented by the offspring of the first generation, that does not have a previous generation. A fresh population of p individuals is randomly generated, except a single individual generated using the greedy heuristic. The mutation and/or generation by crossover could result in an encoding violating the invariant. In order to restore the validity of the invariant, the fix function is applied to each encoding, as detailed in Algorithm 11. Given the encoding of an individual, namely ind, and the encodings of all the cycles, namely $cyclesEnc$, for each unbroken cycle κ (test at line 3) the function randomly chooses one of the mappings traversed by κ (line 4), and then adds it to ind by updating its encoding (line 5).

(vii). In our solution, a multi-strategy termination policy is employed. The first strategy is a maximum number of iterations, namely i, while the second one is the best global solution stagnation detection. When the first of these two conditions is met, the computation is stopped. Specifically, best individual cost stagnation condition is met when no improvements in the best solution occur after a fixed amount of generations. When one of the terminating conditions is satisfied, the best global individual (or one of them, if not unique) among all the iterations is returned.

Example 5.19 provides an example of the encoding used by the GA-based heuristic.

Example 5.19. Consider the SCC shown in Figure 5.4, where two global cycles exist, the unsafe cycle $\kappa^s = [v_c, v_b, v_a, v_d, v_c]$ and the safe cycle $\kappa^u = [v_c, v_b, v_a, v_d, v_e, v_f, v_c]$. Suppose that a mapping ordering r such that $r((v_a, v_d, \_)) = 0$, $r((v_d, v_c, \_)) = 1$, and $r((v_f, v_c, \_)) = 2$, is chosen. At this point $\langle \kappa^s \rangle$, the binary encoding of κ^s, is equal to 110, while $\langle \kappa^u \rangle = 101$. The

Algorithm 12 createDigraph Function

```
 1: function createDigraph(Ontology O_1, Ontology O_2, Alignment M) : Digraph
 2:     V ← N_C(O_1) ⊎ N_C(O_2)
 3:     A ← ∅
 4:     for ax ∈ Axioms(O_i), with i ∈ {1,2} do
 5:         if ax = B ⊑ C and B, C ∈ N_C(O_i) then
 6:             A ← A ∪ {(B, C, 1)}
 7:         end if
 8:         if ax = B ⊒ C and B, C ∈ N_C(O_i) then
 9:             A ← A ∪ {(C, B, 1)}
10:         end if
11:         if ax = B ≡ C and B, C ∈ N_C(O_i) then
12:             A ← A ∪ {(B, C, 1), (C, B, 1)}
13:         end if
14:     end for
15:     for each ⟨B, C, r, w⟩ in M do
16:         if ax = B ⊑ C then
17:             A ← A ∪ {(B, C, w)}
18:         end if
19:         if ax = B ⊒ C then
20:             A ← A ∪ {(C, B, w)}
21:         end if
22:         if ax = B ≡ C then
23:             A ← A ∪ {(B, C, w), (C, B, w)}
24:         end if
25:     end for
26:     return (V, A)
27: end function
```

encoding of the diagnosis $\{(v_a, v_d, \_)\}$ is equal to 100, that of diagnosis $\{(v_d, v_c, \_), (v_f, v_c, \_)\}$ is equal to 011. $\Diamond$

5.4.3 CycleBreaker Algorithm

In this section we detail the two variants of CycleBreaker algorithm, along with their relevant components.

CreateDigraph Function. A core component of CycleBreaker algorithm is the computation of the graph representation associated with a pair of input ontologies and an alignment between them, as defined in Definition 5.2. Algorithm 12 details the createDigraph function, used to create such graph representation. More formally, it builds a digraph G representing the aligned ontology associated with the input ontologies $\mathcal{O}_1$, $\mathcal{O}_2$ and alignment $\mathcal{M}$. In accordance with Definition 5.2, the vertices of this graph are the named concepts of the two ontologies, and its arcs are the axioms/mappings involving them (*for* loops at lines 4–14, 15–25, respectively).

The CycleBreaker algorithm comes in two flavours, detailed in Algorithm 13 and Algorithm 14, respectively. In both versions the input consists of two ontologies, namely $\mathcal{O}_1$ and $\mathcal{O}_2$, and an alignment $\mathcal{M}$ between them. The graph representation G of the aligned ontology w.r.t. $\mathcal{O}_1$, $\mathcal{O}_2$

Algorithm 13 CycleBreaker Algorithm

```
1:  function CycleBreaker(Ontology O_1, Ontology O_2, Alignment M) : Diagnosis
2:      G ← createDigraph(O_1, O_2, M)
3:      SCCS_i ← tarjan(G, O_i)
4:      globalSCCs ← tarjan(G)
5:      mappings ← ∅                                   ▷ Map of SCCs' mappings
6:      cycles ← ∅                                     ▷ Map of SCCs' cycles
7:      Δ ← ∅                                          ▷ Diagnosis for M
8:      for each S in globalSCCs do
9:          if ¬ ⋀_{i=1}^{2} Π_{O_i}(S) ∈ SCCs_i then
10:             mappings(S) ← extractMappings(S)
11:             if solver is not based on ASP then
12:                 cycles(S) ← johnson(S)
13:             end if
14:             Δ ← Δ ∪ solver(S)
15:         end if
16:     end for
17:     return Δ
18: end function
```

and $\mathcal{M}$, is built by means of the createDigraph function (line 2). The SCCs of the input ontologies (line 3) and that of the aligned ontology (line 4) are computed, by means of the *Tarjan's* algorithm. The algorithm then detects which SCCs of the aligned ontology are problematic. By Definition 5.10, it suffices to test if at least one of the two projections on the input ontologies of each considered SCC is not a local SCC (line 9 for Algorithm 13, line 10 for Algorithm 14). To deal with problematic SCCs, the two variants adopt a different strategy, detailed in what follows:

Algorithm 13 In the CycleBreaker algorithm variant detailed in Algorithm 13, the nonconservative (resp. conservative) diagnosis for the current SCC, namely S, is computed by the solver function, that runs an *ASP* solver on an *ASP* program representing an instance of the *MAP-WFES* (resp. *CMAP-WFES*) problem associated with S. The diagnosis for S is then obtained, using a translation, from the solution of the aforementioned problem, if any. In place of the *ASP* program for *CMAP-WFES*, it is also possible to compute an approximation (see Section 5.4.1), in order to guarantee diagnosis existence. Theorem 5.1 and Proposition 5.12 detail, respectively, the correctness of the detection and the solution computation, as described above. Consider that, alternatively, the greedy heuristic and the GA-based heuristic can be employed as concrete implementations of the solver function. Given that both heuristics require as input the distinct minimal cycles in order to compute a diagnosis, these cycles are computed (for each problematic SCC) by means of the *Johnson's* algorithm (line 12).

Algorithm 14 Algorithm 14 keeps track of the problematic SCCs (line 11) and their mappings (line 12). Then all the problematic mappings presenting the same vertex as the source (resp. target) element, except the one with the highest confidence value (line 15), are removed. In this way, a 1-1 filtering step is applied to the original alignment, but restrained to the problematic SCCs. At this point the diagnosis computation totally corresponds to that of Algorithm 13.

Algorithm 14 CycleBreaker Algorithm (Multiple Occurrences Filtering)

```
1:  function CycleBreaker(Ontology O_1, Ontology O_2, Alignment M) : Diagnosis
2:      G ← createDigraph(O_1, O_2, M)
3:      SCCS_i ← tarjan(G, O_i)
4:      globalSCCs ← tarjan(G)
5:      mappings ← ∅                                   ▷ Map of SCCs' mappings
6:      problSCCs ← ∅                                  ▷ Set of problematic SCCs
7:      Δ ← ∅
8:      problMappings ← ∅                              ▷ Set of problematic mappings
9:      for each S in globalSCCs do
10:         if ⋀_{i=1}^{2} Π_{O_i}(S) ∈ SCCs_i then
11:             problSCCs ← problSCCs ∪ {S}
12:             problMappings ← problMappings ∪ extractMappings(S)
13:         end if
14:     end for
15:     filterMultipleOccurrences(G, M, problMappings)   ▷ Mappings are removed from M and G
16:     for each S in problSCCs do
17:         if solver is not based on ASP then
18:             cycles(S) ← johnson(S)
19:         end if
20:         Δ ← Δ ∪ solver(S)
21:     end for
22:     return Δ
23: end function
```

In both variants, the global diagnosis results from the union of the local diagnoses, that is, the diagnoses of the single SCCs. The correctness proof of such composition is given in Proposition 5.13.

The matching phase between multiple mappings sharing the same source (resp. target) node is obtained through the filterMultipleOccurrences function, detailed in Algorithm 15. First, a list of pairs is computed (lines 4–11), having as first component the hash signature of the source (resp. target) node of each mapping, and the mapping itself. The list length is equal to the number of problematic mappings. The list is then sorted on the first component of the pair, then on the mapping confidence in descending order (line 12). Thanks to the sorting, we are guaranteed that, using a traversal of this list (lines 14–25), a single auxiliary variable allows us to keep track of the current element (already inserted) and to discard all the others sharing the same source (resp. target) node. Sorting on mapping confidence also guarantees to keep the mapping with the highest confidence value.

As already discussed, the filterMultipleOccurrences function aims at heuristically computing a 1-1 filtering, that can be optionally used to lower the runtime of the diagnosis computation for problematic SCCs, at the cost of obtaining a suboptimal diagnosis. In Listing 5.5, an alternative method for the 1-1 filtering step, based on *ASP*, is shown. Differently from filterMultipleOccurrences, the *ASP*-based program computes an optimal filtering. Nonetheless, the diagnosis obtained after this filtering will be suboptimal, due to unnecessary mapping removals, not related to equivalence violations. In what follows, we briefly describe the rules composing the *ASP* program:

Algorithm 15 Multiple Occurrences Filtering Function

```
1: function filterMultipleOccurrences(Digraph G, Alignment M, Alignment problM)
2:     for each first in {true, false} do                ▷ Filterings on source node, then on target node
3:         L ← ∅                                         ▷ Empty list created
4:         for each m ← (s, t) in M ∩ problM do
5:             if first then
6:                 h ← hash(s)
7:             else
8:                 h ← hash(t)
9:             end if
10:            L.add(⟨h, m⟩)
11:        end for
12:        sort(L)                                       ▷ sort on first component, then on mapping confidence
13:        tmpHash ← λ
14:        for each ⟨h, m⟩ in L do
15:            if tmpHash ≠ λ then
16:                if tmpHash = h then
17:                    G.remove(m)
18:                    M.remove(m)
19:                else
20:                    tmpHash ← h
21:                end if
22:            else
23:                tmpHash ← h
24:            end if
25:        end for
26:    end for
27: end function
```

r1-r2 : rule $r1$ (resp. $r2$) defines a clash on source (resp. target) element as a pair of at least two mappings sharing the same source (resp. target) element,

r3 : each edge is either removed or not removed,

r4 : valid models remove only mappings,

r5-r6 : these two rules forbid clashes (on source and target elements, respectively) for valid models,

r7 : the optimization objective is defined as the minimization of the total weight of removed mappings,

r8 : this rule restricts the model output to "removed" predicate only.

An empirical comparison between the two 1-1 filtering methods is provided in Section 5.5.7.

As experimentally verified in Section 5.5.2, the best variant of CycleBreaker algorithm is that based on *ASP*, despite the total complexity is dominated by that of *MAP-WFES*/*CMAP-WFES*, that are *NP*-hard problems. This is motivated by the use of an optimized *ASP* solver that performs well in practice. The interested reader can refer to Section C.2, Appendix C, where the temporal complexity of the different variants of the CycleBreaker algorithm, as well as of its subcomponents, is investigated.

```
% r1 - clash on source entity
clash(edge(X,Y,C),edge(X,Z,D)) :- edge(X,Y,C), mapping(edge(X,Y,C)), edge(X,Z,D),
    mapping(edge(X,Z,D)), not_removed(edge(X,Y,C)), not_removed(edge(X,Z,D)), Y!=Z, X!=Y, X!=Z
    .
% r2 - clash on target entity
clash(edge(X,Y,C), edge(Z,Y,D)) :- edge(X,Y,C), mapping(edge(X,Y,C)), edge(Z,Y,D),
    mapping(edge(Z,Y,D)), not_removed(edge(X,Y,C,1)), not_removed(edge(Z,Y,D,1)), Y!=Z, X!=Y,
    X!=Z .
% r3 - an edge is either removed or not removed
not_removed(edge(X,Y,C)) | removed(edge(X,Y,C)) :- edge(X,Y,C), X!=Y .
% r4 - removed edges must be mappings
not_removed(edge(X,Y,C)) :- edge(X,Y,C), not mapping(edge(X,Y,C)), X!=Y.
% r5-r6 - there must be no clashes
:- clash(edge(X,Y,C), edge(X,Z,D)), edge(X,Y,C), mapping(edge(X,Y,C)), edge(X,Z,D),
    mapping(edge(X,Z,D)), Y!=Z, X!=Y, X!=Z .
:- clash(edge(X,Y,C), edge(Z,Y,D)), edge(X,Y,C), mapping(edge(X,Y,C)), edge(Z,Y,D),
    mapping(edge(Z,Y,D)), Y!=Z, X!=Y, X!=Z .
% r7 - optimization
#minimize [ removed(edge(X,Y,C)) = C ] .
% r8 - model output
#hide .
#show removed/1.
```

Listing 5.5: *ASP* program for computing a 1-1 version of an input alignment.

In Table 5.2 we provide a summarization of the different subcomponents of the different CycleBreaker algorithm variants, along with their purpose, if the technique is approximated or not, and the dependencies from other components, if any.

ID	Method	Purpose	Approximated?	Requires	Definition
1	createDigraph Function	Graph Encoding	–	–	Algorithm 12
2	Tarjan's Algorithm	SCCs Detection	NO	1	Definition 4.38
3	Johnson's Algorithm	Cycles Detection	NO	1,2	Definition 4.39
4	1-1 Filter Heuristic	1-1 Filtering	YES	1,2	Algorithm 15
5	1-1 Filter *ASP*	1-1 Filtering	NO	1,2	Listing 5.5
6	Greedy Diagnosis	Nonconservative Diagnosis	YES	1,2,3	Algorithm 10
7	GA-based Diagnosis	Nonconservative Diagnosis	YES	1,2,3	Section 5.4.2
8	Nonconservative *ASP*	Nonconservative Diagnosis	NO	1,2	Listing 5.2
9	Conservative *ASP*	Conservative Diagnosis	NO	1,2	Listing 5.3
10	Approximated Conservative *ASP* 1	Conservative Diagnosis	YES	1,2	Listing 5.4
11	Approximated Conservative *ASP* 2	Conservative Diagnosis	YES	1,2	Listing C.1

Table 5.2: Relevant components of CycleBreaker algorithm.

5.5 Experimental Evaluation

This section experimentally evaluates CycleBreaker algorithm under different dimensions of interest. In Section 5.5.1 we review the relevant details about our prototype, in Section 5.5.2 we compare the performance of the different solvers (and their variants) as well as the quality of the computed diagnosis (*i.e.*, their weights). In Section 5.5.3 we measure the amount of (detected)

equivalence violations in the alignments produced by state of the art matchers, using the well-known *OAEI* dataset (2012-2014) [AEE$^+$12, CDE$^+$13, DEE$^+$14]. Section 5.5.4 extends the analysis of the previous section to two matchers dealing (directly and indirectly, respectively) with the conservativity principle w.r.t. equivalence, that is ASMOV and Lily matchers. Section 5.5.5 aims at investigating the impact of our diagnosis approach on classic ontology alignment quality metrics (*i.e.*, precision, recall and f-measure) w.r.t. the reference alignments provided by *OAEI* organizers. In Section 5.5.6 we investigate how the different subtasks composing the computation of an alignment diagnosis impact on the actual time required by our prototype, at varying characteristics for alignments and ontologies. Section 5.5.7 compares the performance and solution quality (*i.e.*, the total weight of the discarded mappings) of an optimal solution for the 1-1 alignment extraction, and that of the heuristic detailed in Section 5.4.3, Algorithm 15. In Section 5.5.8 we analyze (an extract of) a set of manually curated alignments (*i.e.*, the *UMLS* alignment, 2012, 2013 and 2014 versions). In Section 5.5.9 we discuss manual debugging activity, and how CycleBreaker's prototype may increase efficiency and help improving diagnosis effectiveness. When not explicitly stated, the experiments presented throughout the section are runned over the *OAEI* 2012-2014 dataset,[8] plus additional alignments for Lily (2007) and ASMOV (from 2008 to 2010). As a general remark, we stress that the used dataset has a relatively small size, because only 4 of the available tracks of OAEI provided suitable alignments for our analysis. Due to the size of the available dataset, and to the high variability of the characteristics of the input ontologies and matchers, the standard deviation of our measurements and features computation is not negligible. For this reason, the experimental evaluation proposed in these sections has the aim of strengthening or weakening hypotheses using statistical trends, rather than empirically formulating new ones by inductive reasoning.

5.5.1 Implementation Details

In this section we discuss the relevant details concerning our implementation. The prototype is mainly written in *Java*, it only relies on *Clingo* 3.0.5 [GKK$^+$11] (based, internally, on *Clasp* solver) for executing the *ASP* programs. Given the computational hardness of the different search problems encoded as *ASP* programs, we use a timeout value, after which the *ASP* solver is interrupted. When *Clingo* 3.0.5 is not able to reach the *optimum*, it prints each stable model with a better optimization value w.r.t. all the previously computed ones. In this way, if the timeout is reached, we can use the last produced model (*i.e.*, the best one computed before reaching the timeout) as our diagnosis. When not explicitly specified, the experiments of this section use a timeout value equal to 60 seconds.

OWL-API 3.5.1 [HB11] is used for handling the aligned ontologies, as well as for computing their classification by means of *HermiT* 1.3.8 reasoner[9] [MSH09]. The alignments are loaded using

[8] `http://oaei.ontologymatching.org`

[9] After a timeout of 3 minutes *ELK* 0.4.1 reasoner [KKS11] was used, and this only happened for *SNOMED*

the *Alignment API* 4.7 [DEST11] and a custom RDF parser based on *Jena* library. A fraction of the dataset used throughout our experiments was not compatible with the *Alignment API* 4.7 but it was possible to load it with the custom parser. However, a small percentage (around 1% of the alignments) was neither compatible with the custom parser, and for this reason it has not been considered in the experimental evaluation.

The test environment consists of a desktop computer equipped with 32*GB DDR*3 *RAM* at 1333*MHz*, and an *AMD Fusion FX 4350* (quad-core, each running at 4.2*GHz*) as *CPU*. The dataset is stored on a 128*GB SSD*, where the operating system (*Ubuntu* 12.04, 64-bit version) is installed.

All the results involving stochastic methods or time measurements are averaged over multiple repetitions in order to increase the statistical significance of the experimental results.

5.5.2 Diagnosis Computation Performance

This section is devoted to the comparison of the quality (in terms of diagnosis weight) and temporal efficiency of all the diagnosis computation techniques available in the CycleBreaker algorithm. The SCCs are a subset (around 30% of the total) of those in the full dataset, restricted in order to obtain a reasonable runtime despite the multiple repetitions and the number of different techniques (some of them extremely costly in terms of running time). We compared nonconservative *ASP* program with: *(i)* conservative *ASP* program, *(ii)* the GA-based heuristic, *(iii)* the greedy heuristic, *(iv) ASP* program for nonconservative diagnoses on 1-1 alignments (computed heuristically). Heuristics (ii) and (iii), in order to be applicable to a problematic SCC, require the computation of the elementary cycles of the SCC, and therefore the cycle computation time (using *Johnson's* algorithm), is added to their computational time.

By comparing the solution cost we can deduce that, in the selected SCCs, there were no nontrivial safe cycles to preserve, and therefore the two *ASP* variants coincide. However, the runtime is different, and computing nonconservative diagnoses is faster. The explanation relies in the additional constraints enforced by the conservative variant, that requires additional computation time for the *ASP* solver to compute the solution.

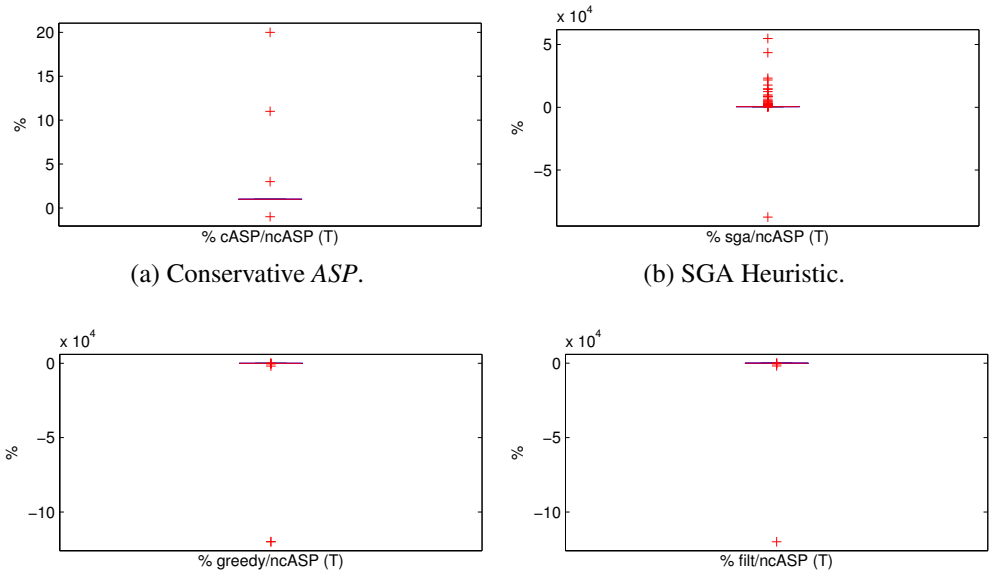
From the experimental results it is also evident that the two proposed heuristics are dominated by the *ASP* programs both in terms of temporal performance, and in terms of diagnosis minimality. In addition, for SCCs with more than ~ 120 mappings, we experienced some $OutOfMemoryException$ (due to the consumption of all the available heap memory for the running *Java Virtual Machine*), despite a memory allocation size of 13GB for the test process. In such cases, the number of distinct elementary cycles exceeded 5 millions. On the contrary, the *ASP* solver never required more than ~ 300MB, and despite the reach of the temporal timeout, it

ontology.

was always able to produce a (possibly suboptimal) diagnosis. This situation, however, affects less than $\%1$ of the total number of SCCs. These cases correspond to the outliers below the median in Figure 5.8.

For what concerns diagnosis minimality, Figure 5.9 shows that conservative and nonconservative diagnoses coincide, and that a heuristic may compute suboptimal diagnoses (with up to the 15% of additional weight w.r.t. the optimal diagnosis).

In the remainder of the experiments, we focus on the best performing diagnosis computation technique, that is, the nonconservative (variant of the) *ASP* program.



(a) Conservative *ASP*.

(b) SGA Heuristic.

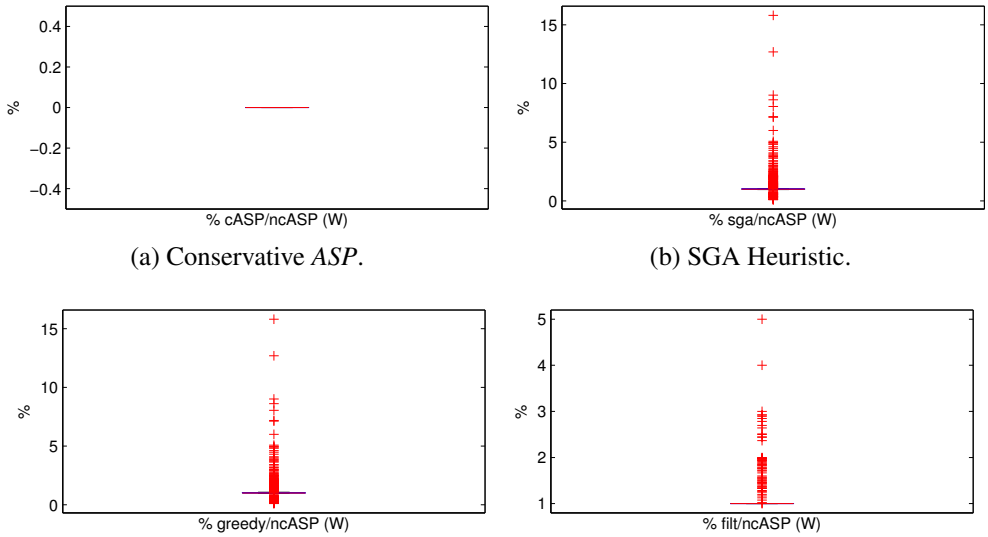
(c) Greedy Heuristic.

(d) 1-1 filtering.

Figure 5.8: Time comparison w.r.t. nonconservative diagnosis computation.

5.5.3 Analysis of Ontology Matchers Alignments

For testing the effect of our repair algorithm we used the data of the *OAEI* dataset (years 2012–2014), in particular the *anatomy*, *largebio*, *conference*, and *library* tracks. Similarly to [Mei11], we introduce a measure for computing the "degree of problematicity" of an alignment, related to the violations of conservativity principle w.r.t. equivalence that appears in it. This measure expresses the fraction of mappings belonging to problematic SCCs over the total alignment cardinality. Intuitively, this measure denotes the (upperbound) of an alignment that should be removed in order to solve any violation to the conservativity principle w.r.t. equivalence, in a context where a debugging algorithm is not available.



(a) Conservative *ASP*.

(b) SGA Heuristic.

(c) Greedy Heuristic.

(d) 1-1 filtering.

Figure 5.9: Diagnosis weight comparison w.r.t. nonconservative diagnosis computation.

More formally, given an alignment $\mathcal{M}$ between two input ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, and a diagnosis Δ for $\mathcal{M}$, the *degree of problematicity* for the alignment and the corresponding input ontologies is defined as $PrCP_{\equiv}(\mathcal{M}, \mathcal{O}_1, \mathcal{O}_2) = |\mathcal{M}^P|/|\mathcal{M}|$, where $\mathcal{M}^P$ is the set of problematic mappings (see Definition 5.11).

Table 5.3 and Table 5.4 summarize relevant features of the dataset and analysis results. All the results computed after a grouping operation presents the format *mean* (*standard deviation*) in place of a single aggregate value.

Effectiveness Measures. Data column **(i)** is the average percentage of mappings removed by the diagnosis algorithm w.r.t. the cardinality of the problematic mappings. This quantity may be used as a comparison between a minimal diagnosis and a bruteforce approach removing all the problematic mappings. This last scenario applies where no methods eligible to compute a diagnosis are present. Column **(ii)** represents the fraction of problematic mappings w.r.t. the total alignment, and, as already discussed, it represents the degree of problematicity of an alignment.

Column **(iii)** is the average impact of a minimal diagnosis on the alignment (that is, the minimum fraction of mappings that necessarily needs to be removed from the alignment to avoid violations of the conservativity principle). Column **(iv)** presents the percentage of elements with 1-1 mappings only (that is, elements not occurring multiple times as source or target element in different mappings). Column **(v)** is the average alignment size. Column **(vi)** is the fraction of vertices belonging to problematic SCCs, and it is useful to measure the average size of problem-

atic SCCs, and as an approximate measure of the logical average impact of the violations of the conservativity principle, for the considered alignments. Column **(vii)** is the number of problematic SCCs. Column **(viii)** shows the total time spent by the *ASP* solver for diagnosis computation. Column **(ix)** shows the average percentage of optimal diagnoses. Suboptimal diagnoses may be used in place of minimal ones when the *ASP* solver reaches the timeout. In such cases the best diagnosis computed so far (possibly suboptimal) is returned. Column **(x)** shows the number of (problematic) alignments considered for computing the actual row, while column **(xi)** is the total number of alignments available for the considered grouping.

The difference between column **(xi)** and columns **(x)** are nonproblematic alignments, that are excluded from the analysis (given their lack of problematic SCCs). The information on the number of used alignments is particularily useful given the high difference of available alignments between various matchers and tracks, in the *OAEI* dataset.

The considerations so far relate to the effectiveness of the approach (in particular, diagnoses minimality and their impact on alignments).

Efficiency Measures. The percentage of optimal diagnoses (*i.e.*, column **(ix)**), is an indicator of the efficient and tractable diagnosis computation for conservativity principle violation w.r.t. equivalence, enabled by the employed techniques. Performance has been already discussed in greater details in Section 5.5.2, where the *ASP*-based solution has been compared with the other heuristics proposed in Section 5.4 (*i.e.*, GA-based and greedy heuristics).

From Table 5.3 we can observe that the degree of problematicity ranges sensibly from matcher to matcher. A positive correlation exists between the percentage of elements with 1-1 mappings only, and the degree of problematicity. From columns **(i-iii)** it emerges that minimal diagnoses save, on average, from ~62% to ~99% of the problematic mappings, and their impact on alignments does not exceed 12%, except for the four worst-performing matchers (namely Ase, Autom, Hertuda and RIMOM).

Table 5.4 confirms the positive correlation between the percentage of elements with 1-1 mappings only, and the degree of problematicity. In addition, Table 5.4 shows that a positive correlation exists between the number of entities in the input ontologies (shown in Table 5.5) and the degree of problematicity. In particular, the number of nonproblematic alignments (the difference between column **(xi)** and column **(x)**) is ~84% for *conference* track, while it is ~35% for *anatomy* track, ~7% for *largebio* (small) track, ~12% for *library* and 0% for *largebio* (big) track. Different groupings for the same experimental values are provided in [Sol15], Section 1.1.

Effectiveness w.r.t. Unsolved Violations. Tables 5.6 and 5.7,[10] restricted to the *OAEI* 2012–2014 dataset, present the size of the input ontologies (columns **i** and **ii**), the size of the input alignment (column **iii**), the number of all the equivalence violations (column **iv**), the diagnosis size (column **v**) and the number of unsolved equivalence violations after diagnosis application (column **vi**).

The tables confirm the effectiveness of our approach, that is able to fully remove all equivalence violations for the *anatomy* and *library* tracks. Despite the incompleteness of the method, for the other tracks at most an average of approximately 4% of the violations are not solved.

5.5.4 Comparison of State of the Art Repair Systems

This section aims at experimentally assessing the effectiveness of the repair strategies of Lily and ASMOV, described in Section 4.6.4.

While Lily claims to address (a subcase of) equivalence violations, it is not possible to directly compare our implementation with theirs, given that Lily does not implement any kind of correction, and the detected cycles are only reported to the user. ASMOV matcher, in addition, does not directly support detection or correction for conservativity principle violations w.r.t. equivalence. Moreover, ASMOV is a commercial product and no trial versions are available for testing purposes.

Therefore, in order to assess the effectiveness of the proposed repair strategies, we analyze the presence of problematic mappings in the alignment produced by Lily and ASMOV for the *anatomy* and *conference* tracks of OAEI 2007[11] and OAEI 2008–2010[12], respectively. Unfortunately, neither *library* nor *largebio* track were present before OAEI 2011. From the experimental data shown in Table 5.3 it is evident that conservativity principle violations are not avoided by both matchers, even if they are not among the worst-performing ones w.r.t. the degree of problematicity, thanks to the high percentage of 1-1 mappings. In our dataset many alignments produced by ASMOV and Lily are not problematic. This is due to the high percentage of alignments for ontologies belonging to the *conference* track. This results, like for the other matchers, seem to be more influenced by the size of the input ontologies than by the effect of techniques preventing conservativity principle violations (as discussed in Section 5.5.3). As a general trend, we have that the number of violations is positively correlated with the size of the alignment. This result is not surprising, given that violations are represented by a subclass of the cycles of the aligned ontology that involve mappings. The precise number of cycles is influenced by nodes connectivity, but a positive correlation between the number of cycles and the number of arcs in a graph exists.

[10] Note that these tables do not exclude alignments without equivalence violations, in order to be comparable with the tables of the remaining chapters.

[11] Available at `https://sites.google.com/site/ontomappinglab/oaei2007`

[12] Available at `http://infotechsoft.com/products/asmov.aspx`

System	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)
	% D/PM	% PM/M	% D/M	% 1-1M	$\|\mathcal{M}\|$	% VtxSCC/Vtx	probSCC	ASP (s)	% OptDiag	$\#\mathcal{M}$	$\#\text{Tot}\mathcal{M}$
aml	25.78(5.87)	18.41(21.19)	5.56(8.77)	81.91(20.73)	12,547.25(8,434.16)	5.66(8.73)	378.38(379.58)	9.66(29.76)	99.99(0.04)	16	59
amlbk	24.41(2.12)	6.84(7.61)	1.65(1.78)	93.32(7.55)	14,536.67(8,891.46)	1.84(2.84)	199(290.32)	1.41(1.53)	100(0)	6	28
amlbkr	23.8(2.29)	5.79(6.39)	1.38(1.49)	94.36(6.33)	13,943.67(8,532.19)	1.5(2.33)	163.67(235.25)	1.09(1.17)	100(0)	6	6
amlbku	26.44(0.95)	32.61(7.09)	8.67(2.19)	67.49(7.05)	19,944.33(13,448.49)	7.88(6.03)	1,401.67(1,121.21)	10.18(10.44)	100(0)	6	6
amlbkur	26.3(1.13)	29.69(6.39)	7.86(2.04)	70.4(6.34)	18,906.67(12,874.03)	6.89(5.26)	1,264.33(1,034.97)	9.27(9.71)	100(0)	6	6
amlr	23.9(2.52)	4.88(5.63)	1.16(1.29)	95.27(5.57)	13,358.67(8,405.04)	1.27(1.95)	140.33(213.5)	0.95(1.04)	100(0)	6	6
aroma	28.44(14.23)	20.18(19.96)	8.04(17.91)	80.11(19.69)	4,462.6(9,093.43)	4.78(9.83)	64.1(124.72)	12.4(42.24)	99.8(0.81)	20	28
ase	35.22(5.28)	66.52(17.75)	23.82(7.43)	33.76(17.64)	49.89(19.55)	15.44(7.62)	3.94(1.8)	0.64(2.62)	100(0)	18	21
asmov	25.26(1.15)	9.29(4.34)	2.36(1.09)	97.97(3.66)	1,000.41(1,386.52)	3.35(1.76)	12(19.6)	0.03(0.03)	100(0)	27	312
autom	33.61(4.81)	52(20.15)	18.14(8.17)	48.14(20.42)	927(1,794.02)	5.08(1.18)	29.5(57)	15.07(30.12)	99.78(0.43)	4	22
cidercl	22.8(3.23)	14.94(8.33)	3.32(1.62)	100(0)	346.18(1,020.19)	3.91(3)	3(5.67)	0.01(0.01)	100(0)	11	22
codi	21.93(4.34)	0.88(1.03)	0.17(0.19)	99.88(0.16)	4,588(2,802.97)	0.41(0.49)	10.5(13.44)	0.04(0.04)	100(0)	2	23
cromatcher	25(0)	6.55(2.99)	1.64(0.75)	100(0)	68.67(24.85)	2.61(0.99)	1(0)	0	100(0)	3	15
gomma	26.72(8.16)	16.06(17.38)	5.44(10.16)	84.12(16.89)	10,561(7,903.48)	4.14(7.71)	338.69(395.84)	17.3(36.62)	99.98(0.05)	16	37
gommabk	24.8(1.13)	27.19(15.55)	6.81(4)	72.85(15.48)	16,185.43(11,273.43)	5.9(3.71)	1,058.86(1,106.38)	14.62(28.1)	99.99(0.04)	7	7
gommasbk	25.01(1.32)	30.12(15.26)	7.6(3.96)	69.97(15.21)	18,492.67(10,722.76)	5.85(4.25)	1,237(1,126.19)	30.64(46.35)	99.98(0.06)	6	6
hertuda	45.19(15.17)	67.38(13.91)	31.85(17.69)	33.32(12.86)	8,917.33(2,976.31)	20.2(17.12)	379.67(120)	127.22(50.88)	98.39(1.96)	3	50
hotmatch	25(0)	0.13(0.06)	0.03(0.01)	100(0)	4,419(199.4)	0.05(0.04)	1.5(0.71)	0.04(0.02)	100(0)	2	50
iama	27.37(1.12)	7.78(5.15)	2.15(1.5)	92.46(5.05)	6,837.43(7,204.2)	0.43(0.33)	110.14(120.72)	1.01(1.22)	100(0)	7	29
lily	23.33(7.24)	15.55(13.2)	3.46(2.6)	100(0)	273.22(858.71)	8.36(9.27)	1.91(2.3)	0.02(0.03)	100(0)	46	108
logmap	24.48(0.62)	21.73(11)	5.36(2.84)	78.4(10.87)	10,427.71(9,014.58)	5.05(4.11)	522.86(486.51)	3.12(4.01)	100(0)	28	88
logmap2noe	24.35(0.48)	19.77(3.42)	4.81(0.84)	80.28(3.49)	14,836.83(9,642.32)	4.2(3.06)	687.33(522.25)	4.38(4.57)	100(0)	6	6
logmapbio	24.42(0.37)	25.69(6.34)	6.28(1.59)	74.43(6.3)	13,713.86(9,793.49)	5.42(3.98)	766.14(577.35)	5.49(5.45)	100(0)	7	7
logmapbk	24.38(0.5)	20.47(3.94)	4.99(0.98)	79.71(3.97)	14,654(9,535.63)	4.12(3.39)	713.5(561.84)	5(4.82)	100(0)	6	6
logmapc	25.05(0.18)	2.17(1.85)	0.55(0.46)	92.61(6.75)	8,427.71(5,797.1)	0.29(0.3)	57.57(77.54)	0.67(0.91)	100(0)	7	29
logmaplt	27.3(5.61)	25.79(20.7)	7.97(8.51)	74.43(20.34)	8,533.93(8,676.79)	5.24(7.49)	493.11(574.53)	26.03(46.44)	99.96(0.08)	27	87
maasmatch	22.34(13.26)	8.55(4.73)	1.91(2.04)	100(0)	1,426.78(3,357.42)	5.59(3.58)	5.37(13.94)	6.85(27.83)	99.55(1.65)	41	71
mapsss	24.2(0.69)	1.96(1.54)	0.47(0.37)	100(0)	6,481.67(6,495.58)	0.42(0.29)	28(26.83)	0.15(0.17)	100(0)	6	49
medley	25.93(3.03)	38.88(13.28)	10.01(3.52)	63.65(12.62)	71.3(34.98)	14.95(6.77)	5.7(3.5)	0.01(0.01)	100(0)	20	21
odgoms	26.05(2.77)	14.69(20.76)	4.28(6.51)	85.69(20.67)	6,838.4(3,077.22)	5.3(7.58)	206.2(300.15)	0.57(0.87)	100(0)	5	48
omreasoner	28.68(6.37)	11.93(18.86)	4.22(6.86)	88.14(18.8)	6,695.33(6,534.92)	0.54(0.73)	37.33(31.66)	40.29(69.15)	96.79(5.55)	3	25
optima	27.19(3.8)	18.5(11.83)	5.11(3.55)	83.57(13.75)	599.67(901.3)	2.7(0.8)	23.5(39.97)	0.59(1.24)	100(0)	6	23
rimom	39.35(2.85)	90.03(5.42)	35.49(3.91)	9.97(5.42)	112(47.01)	21.29(7.28)	3.33(1.53)	29.58(29.29)	80(30.89)	21	21
rsdlwb	25(0)	1.17(0)	0.29(0)	100(0)	342(0)	0.03(0)	1(0)	0.03(0)	100(0)	1	24
servomap	24.59(4.27)	19.53(16.68)	4.93(4.42)	80.66(16.69)	7,664.08(9,016.82)	3.37(4.84)	274.25(532.39)	4.72(13.17)	100(0.02)	24	58
servomapl	25.48(0.91)	14.22(9.11)	3.67(2.49)	86.27(8.57)	11,687(8,806.28)	3.41(3.55)	380.62(278.26)	9.68(20.78)	99.97(0.08)	8	29
sphere	26.57(1.19)	6.4(2.47)	1.71(0.68)	93.71(2.48)	9,331.33(7,650.3)	0.68(0.56)	144.83(142.7)	1.31(1.31)	100(0)	6	6
stringsauto	23.66(1.56)	2.18(1.72)	0.51(0.41)	97.98(2.09)	2,714.67(1,129.85)	0.52(0.44)	10.67(7.57)	0.03(0.01)	100(0)	3	24
toast	25(0)	0.3(0)	0.07(0)	100(0)	2,678(0)	0.13(0)	2(0)	0.01(0)	100(0)	1	1
xmap	30.5(11.34)	15.91(27.64)	7.4(16.47)	84.5(27.17)	13,921.5(9,496.49)	6.19(13.24)	235.25(256.39)	23.92(44.69)	99.83(0.41)	8	29
xmapgen	29.05(12.81)	36.33(18.51)	11.97(15.66)	62.3(17.4)	2,607.83(13,616.36)	8.17(10.1)	19.46(50.36)	10.16(39.62)	99.91(0.35)	35	46
xmapsig	26.71(2.9)	25.71(8.3)	6.91(2.53)	71.65(9.18)	840.71(1,353.44)	3.77(1.89)	36.07(60.09)	9.74(24.79)	99.73(0.8)	14	46
yam++	24.24(0.63)	16.94(18.02)	4.05(4.2)	83.21(17.87)	12,169.75(8,326.91)	4.66(7.23)	364.88(297.41)	9.61(20.31)	99.99(0.04)	16	58

Table 5.3: Measures of interest for *OAEI* 2012-2014 dataset plus ASMOV (2008-2010) and Lily (2007). Results for different alignments are grouped by matcher.

Track	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)
	% D/PM	**% PM/M**	**% D/M**	**% 1-1M**	$\lvert\mathcal{M}\rvert$	**% VtxSCC/Vtx**	**probSCC**	**ASP (s)**	**% OptDiag**	**#$\mathcal{M}$**	**#Tot$\mathcal{M}$**
anatomy	24.36(4.15)	6.46(5.83)	1.7(1.79)	93.9(6.18)	2,842.42(676.3)	2.12(1.72)	33.35(28.02)	0.18(0.54)	100(0)	40	62
conference	26.75(6.85)	30.57(26.5)	9.37(10.51)	75.07(29.98)	58.17(37.1)	8.55(8)	2.16(1.98)	2.61(11.85)	98.27(10.52)	243	1,512
largebio_big	25.58(1.45)	17.27(14.41)	4.42(3.75)	82.75(14.34)	13,867.66(9,055.97)	1.2(1.37)	585.26(725.65)	12.61(25.12)	99.98(0.13)	91	91
largebio_small	24.36(3.6)	14.56(11.48)	3.73(3.6)	85.8(11.65)	12,346.4(8,809.6)	4.51(3.73)	400.04(516.74)	9.46(26.31)	99.71(1.24)	113	122
library	38.01(22.42)	45.56(30.2)	20.78(23.78)	55.5(30.17)	9,678.7(14,692.81)	17.44(15.29)	409(309.99)	55.42(67.44)	99.54(1.52)	30	34

Table 5.4: Measures of interest for *OAEI* 2012-2014 dataset plus ASMOV (2008-2010) and Lily (2007). Results for different alignments are grouped by track.

Ontology	Track	# Classes	# Datatype Prop.	# Object Prop.	DL
NCI (Anatomy)	Anatomy	3304	0	2	$\mathcal{S}$
Adult Mouse Anatomy	Anatomy	2744	0	3	$\mathcal{ALE}$
Cmt	Conference	36	10	49	$\mathcal{ALCIN}(\boldsymbol{D})$
Cocus	Conference	55	0	35	$\mathcal{ALCIF}$
Conference	Conference	60	18	46	$\mathcal{ALCHIF}(\boldsymbol{D})$
Confious	Conference	57	5	52	$\mathcal{SHIN}(\boldsymbol{D})$
ConfOf	Conference	38	23	13	$\mathcal{SIN}(\boldsymbol{D})$
Crs	Conference	14	2	15	$\mathcal{ALCIF}(\boldsymbol{D})$
Edas	Conference	104	20	30	$\mathcal{ALCOIN}(\boldsymbol{D})$
Ekaw	Conference	74	0	33	$\mathcal{SHIN}$
Iasted	Conference	140	3	38	$\mathcal{ALCIN}(\boldsymbol{D})$
OpenConf	Conference	62	21	24	$\mathcal{ALCOI}(\boldsymbol{D})$
Linklings	Conference	37	16	31	$\mathcal{SROIQ}(\boldsymbol{D})$
Micro	Conference	32	9	17	$\mathcal{ALCOIN}(\boldsymbol{D})$
MyReview	Conference	39	17	49	$\mathcal{ALCOIN}(\boldsymbol{D})$
Paperdyne	Conference	47	21	61	$\mathcal{ALCHIN}(\boldsymbol{D})$
Pcs	Conference	23	14	24	$\mathcal{ALCIF}(\boldsymbol{D})$
Sigkdd	Conference	49	11	17	$\mathcal{ALEI}(\boldsymbol{D})$
FMA-small (for NCI)	Largebio-small	3696	24	0	$\mathcal{ALCN}(\boldsymbol{D})$
FMA-small (for SNOMED)	Largebio-small	10157	24	0	$\mathcal{ALCN}(\boldsymbol{D})$
FMA-big (for NCI)	Largebio-big	28861	24	0	$\mathcal{ALCN}(\boldsymbol{D})$
FMA-big (for SNOMED)	Largebio-big	50523	24	0	$\mathcal{ALCN}(\boldsymbol{D})$
FMA (whole)	Largebio-big	78988	54	0	$\mathcal{ALCN}(\boldsymbol{D})$
NCI-small (for FMA)	Largebio-small	6488	0	63	$\mathcal{ALC}$
NCI-small (for SNOMED)	Largebio-small	23958	0	82	$\mathcal{ALCH}$
NCI-big (for FMA)	Largebio-big	25591	0	87	$\mathcal{ALCH}$
NCI-big (for SNOMED)	Largebio-big	49795	0	94	$\mathcal{ALCH}$
NCI (whole)	Largebio-big	66724	1	123	$\mathcal{ALCH}(\boldsymbol{D})$
SNOMED-small (for FMA)	Largebio-small	13412	0	18	$\mathcal{ALER}$
SNOMED-small (for NCI)	Largebio-small	51128	0	51	$\mathcal{ALER}$
SNOMED-big (for NCI-FMA)	Largebio-big	122464	1	55	$\mathcal{ALER}$
STW	Library	6575	0	0	$\mathcal{AL}$
TheSoz	Library	8376	0	0	$\mathcal{AL}$

Table 5.5: Ontology statistics for *OAEI* dataset.

Track	i	ii	iii	iv	v	vi
	$\lvert\mathsf{Sig}(\mathcal{O}_1)\rvert$	$\lvert\mathsf{Sig}(\mathcal{O}_2)\rvert$	$\lvert\mathcal{M}\rvert$	eqViol	$\lvert\Delta\rvert$	eqViol
aml	12,224(29,889)	10,806(27,190)	3,462(7,054)	23,359(178,169)	156(509)	3.47(14)
amlbk	12,527(30,946)	10,936(28,172)	3,235(7,146)	56(214)	35(125)	2.61(11)
amlbkr	57,614(45,446)	50,061(44,114)	13,944(8,532)	290(467)	178(257)	15(29)
amlbku	57,614(45,446)	50,061(44,114)	19,944(13,448)	2,144(2,231)	1,521(1,402)	19(30)
amlbkur	57,614(45,446)	50,061(44,114)	18,907(12,874)	2,383(2,470)	1,673(1,508)	19(30)
amlr	57,614(45,446)	50,061(44,114)	13,359(8,405)	240(395)	151(232)	15(28)
aot	391(933)	560(1,501)	692(1,872)	113(439)	53(190)	0.68(2.98)
aotl	382(912)	539(1,469)	110(327)	7.87(19)	5.48(14)	0
aroma	5,540(14,285)	7,336(24,043)	2,042(5,072)	38(99)	28(75)	1.15(5.57)
ase	107(27)	123(44)	47(27)	9.75(18)	3.8(4.35)	0
autom	274(770)	413(1,372)	179(768)	5.68(27)	4.68(22)	0
cidercl	237(576)	270(697)	208(737)	4.86(21)	1(4.14)	0
codi	506(1,433)	618(1,817)	419(1,445)	8.74(41)	0.91(4.17)	0
cromatcher	107(36)	121(46)	81(22)	0.27(1.03)	0.07(0.26)	0
gomma	17,023(33,634)	15,225(31,294)	4,576(7,354)	55,801(338,465)	224(731)	2.84(6.61)
gommabk	38,540(42,323)	35,059(41,568)	16,185(11,274)	1,303(1,344)	914(975)	5.57(9.86)
gommasbk	57,614(45,446)	50,061(44,114)	18,493(10,722)	1,517(1,358)	1,055(993)	6.83(11)
hertuda	1,021(2,411)	1,368(3,245)	1,205(3,056)	154,430(762,715)	446(1,335)	0
hotmatch	1,021(2,411)	1,368(3,245)	661(1,562)	2.44(11)	0.6(2.39)	0
iama	12,321(30,409)	10,848(27,669)	1,722(4,455)	57(171)	32(87)	0.76(2.85)
logmap	11,362(28,458)	10,158(26,202)	3,332(6,999)	258(609)	184(409)	5.39(21)
logmap2noe	44,505(43,018)	40,351(42,874)	14,837(9,642)	1,009(900)	751(593)	25(42)
logmapbio	49,776(46,381)	43,382(43,977)	13,713(9,794)	1,351(1,256)	872(673)	28(52)
logmapbk	57,614(45,446)	50,061(44,114)	14,654(9,536)	1,073(995)	785(645)	28(49)
logmapc	12,321(30,409)	10,848(27,669)	2,126(4,523)	17(53)	14(44)	0.17(0.66)
logmaplt	11,417(28,618)	10,178(26,353)	2,662(6,201)	16,550(87,249)	198(526)	2.79(10)
maasmatch	607(1,629)	831(2,281)	917(2,622)	7,135(59,956)	19(141)	6.04(51)
mapsss	2,258(8,392)	1,832(5,379)	867(3,004)	13(49)	3(11)	1.92(13)
medley	110(31)	121(44)	98(43)	3.62(6.99)	2.52(4.11)	0
odgoms	870(2,306)	1,182(3,133)	777(2,295)	59(362)	30(178)	0
omreasoner	5,857(18,402)	4,539(14,089)	927(2,929)	47(218)	28(130)	0.32(1.6)
ontok2	110(31)	121(44)	18(6.73)	0	0	0
optima	506(1,433)	618(1,817)	179(500)	25(91)	12(39)	0
rimom	110(31)	121(44)	114(48)	32(31)	27(31)	1.67(7.41)
rsdlwb	640(1,547)	866(2,150)	168(473)	0.08(0.41)	0.04(0.2)	0
servomap	10,965(27,944)	9,844(25,912)	3,215(6,862)	1,371(9,139)	118(393)	2.14(8.77)
servomapl	9,609(25,712)	8,839(24,477)	3,233(6,898)	500(1,952)	100(211)	2.34(8.77)
sphere	57,614(45,446)	50,061(44,114)	9,331(7,650)	255(281)	164(165)	7.17(11)
stringsauto	640(1,547)	866(2,150)	358(968)	2.46(7.68)	1.67(5.64)	0
synthesis	110(31)	121(44)	17(6.79)	0	0	0
toast	2,747(0)	3,306(0)	2,678(0)	4(0)	2(0)	0
wesee	371(1,094)	446(1,380)	245(974)	1.07(5.76)	0.36(1.96)	0
wmatch	308(757)	405(1,149)	258(1,037)	16(85)	12(61)	0
xmap	12,321(30,409)	10,848(27,669)	3,852(7,909)	114,705(617,312)	242(1,065)	6.28(24)
xmapgen	473(1,620)	630(2,221)	242(811)	61(256)	16(61)	0.4(2.68)
xmapsig	605(1,838)	799(2,475)	267(823)	45(192)	15(50)	0
yam++	10,965(27,944)	9,844(25,912)	3,375(6,945)	784(4,516)	111(270)	2.47(9.62)

Table 5.6: Measures of interest for *OAEI* 2012-2014 dataset, grouped by matcher.

System	i	ii	iii	iv	v	vi
	$\|\mathsf{Sig}(\mathcal{O}_1)\|$	$\|\mathsf{Sig}(\mathcal{O}_2)\|$	$\|\mathcal{M}\|$	eqViol	$\|\Delta\|$	eqViol
anatomy	2,747(0)	3,306(0)	2,607(816)	137(340)	82(190)	0
conference	110(30)	121(43)	33(35)	1.27(7.74)	0.85(5.83)	0.05(1.16)
largebio_big	18,340(19,857)	13,450(6,995)	11,789(8,729)	716(1,100)	453(661)	18(49)
largebio_small	84,567(30,642)	79,217(33,050)	13,868(9,056)	822(1,070)	580(735)	12(19)
library	6,575(0)	8,376(0)	6,363(3,168)	516,289(1,125,536)	1,401(1,866)	0

Table 5.7: Measures of interest for *OAEI* 2012-2014 dataset, grouped by track.

5.5.5 Repair Effects on Alignment Quality

In this section we investigate the effect of alignment repair (*i.e.*, diagnosis application for removing violations of the conservativity principle). To this aim we employ three standard measures originating from *Information Retrieval*, and used to define the quality of an alignment: precision, recall and f-measure.

Intuitively, the precision of an alignment w.r.t. a reference alignment expresses the fraction of correct mappings belonging to the alignment. Recall is the fraction of correct mappings (that is, belonging to the reference alignment) that also belong to the computed/repaired alignment. The quality of an alignment is defined by a combination of these two measures. In particular, the quality of a system is a trade-off between precision and recall. A standard measure that combines precision and recall is f-measure.

Formally, the measures of interest are defined as follows. Given an alignment $\mathcal{M}$ and a reference alignment $\mathcal{M}^{\mathcal{R}}$, we denote precision as $pr(\mathcal{M}, \mathcal{M}^{\mathcal{R}}) = |\mathcal{M} \cap \mathcal{M}^{\mathcal{R}}|/\mathcal{M}$, recall as $rc(\mathcal{M}, \mathcal{M}^{\mathcal{R}}) = |\mathcal{M} \cap \mathcal{M}^{\mathcal{R}}|/\mathcal{M}^{\mathcal{R}}$ and f-measure as $fm(\mathcal{M}, \mathcal{M}^{\mathcal{R}}) = 2 \cdot pr(\mathcal{M}, \mathcal{M}^{\mathcal{R}}) \cdot rc(\mathcal{M}, \mathcal{M}^{\mathcal{R}})/(pr(\mathcal{M}, \mathcal{M}^{\mathcal{R}}) + rc(\mathcal{M}, \mathcal{M}^{\mathcal{R}}))$.

In Figure 5.10 the average impact on precision, recall and f-measure of the application of diagnoses is shown (first three boxplots). The last two boxplots show the percentage of entities exclusively having 1-1 mappings. The first of these two boxplots presents the percentage of 1-1 for the reference alignment (denoted as 1-1r in the figure), while the second one shows that of the computed alignment (denoted as 1-1m). Note that 1-1 filtering is not applied to the whole alignment, but only on problematic SCCs. For this reason, even when such filtering is applied, the percentage of entities exclusively having 1-1 mappings could be less than 100%.

The results presented in this section are grouped by the analyzed tracks (*anatomy*, *conference*, *largebio* and *library*). The two variants of *largebio* track, *big* (using the whole ontologies) and *small* (using only a subpart of the input ontologies) variants are analyzed separately, given the significant difference in the alignment size produced in the two cases.

The impact of alignment repair (*i.e.*, diagnosis application) is presented as the percentual of gain (that corresponds to a loss for negative values) of each measure computed for a repaired alignment, compared to the same measure computed for the original alignment. In both cases the

measures are computed w.r.t. the original reference alignment (provided by the *OAEI* organizers). Given that we are investigating the effect of alignment repair, we filtered any alignment that does not violate conservativity principle, because its diagnosis is empty and the gain is always zero.

Figure 5.10 shows that the median value for the gain is close to zero, independently from the measure and the considered track (slight gain for precision in *conference* and loss for recall in *library* track). In general, precision tends to increase, recall tends to decrease, as expected for a repair algorithm. The gains for precision and recall result in a slight increase of the resulting f-measure for *conference* track. Instead, for *anatomy*, both variants of *largebio*, and *library* track, the result is substantially invariate.

From the results, a significant impact (neither positive nor negative) of the diagnosis application in the considered measures does not emerge. This is expected, given the average small size of the computed diagnoses (both in terms of weight and number of removed mappings), which tends to remove the least possible knowledge, in accordance with the principle of minimal change.

As a conclusion, despite the logical impact of violations, their repair does not significantly affect the performance of an alignment w.r.t. a reference alignment. Results aggregated by matcher are available in [Sol15], Section 1.2. In Figure 5.11, instead, the repaired alignment originates from a two-step repair. The first step computes, from the original alignment, a 1-1 corresponding alignment, using the heuristic presented in Algorithm 15 and discussed in Section 5.4.3. The second step computes a nonconservative diagnosis, using *ASP*, on the 1-1 alignment obtained at the previous step. The final diagnosis corresponds to the union of the diagnosis and the mappings filtered by the heuristic for extracting the 1-1 alignment.

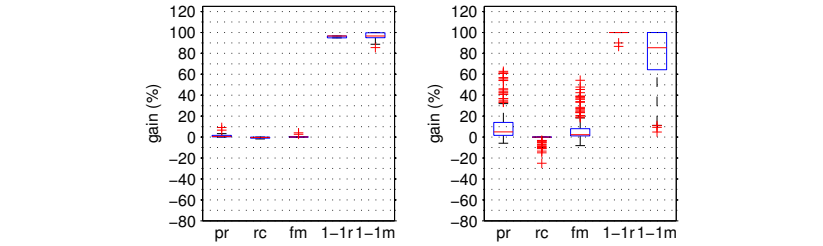
The positive gain on precision and negative one for recall are not surprising. Instead, the average f-measure gain is slightly higher than the corresponding one for the diagnosis without filtering, and this conflicts with the definition of minimal diagnosis.

Two orthogonal explanations are possible for this result. The first one relates to the actual minimality of the computed diagnosis: given the hardness of computing a minimal diagnosis (both conservative and nonconservative), the *ASP* solver could take a significant amount of time to compute the optimal (minimal) solution. For this reason, our prototype employes a timeout for such execution. If the runtime of the *ASP* solver exceeds the timeout, the best answer set (*i.e.*, minimal diagnosis) computed so far is returned as a solution.

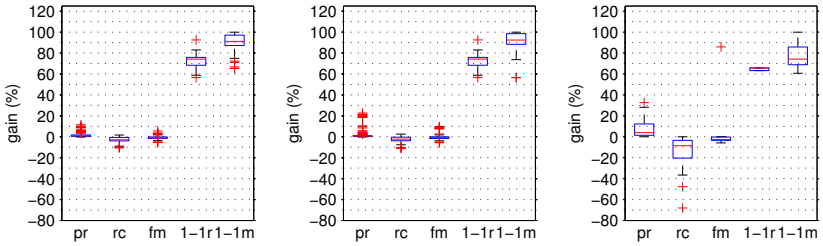
The second explanation, is that, most of the reference alignments tend to be perfect 1-1 alignments (*i.e.*, the amount of multiple correspondences is strongly limited, never more than 25%, as shown in Figure 5.10 and Figure 5.11). An exception to this is the reference alignment for *library* track (1-1 mappings are $\sim 65\%$), and indeed for this track the gain for f-measure is higher for the diagnosis without 1-1 filtering.

Given that, according to the experimental data shown in Table 5.4, almost all the diagnoses are optimal, it is evident that the greatest contribution to the increase of f-measure's gain is given by

the second motivation (that is, reference alignments tend to be (close to) a perfect 1-1 alignment).

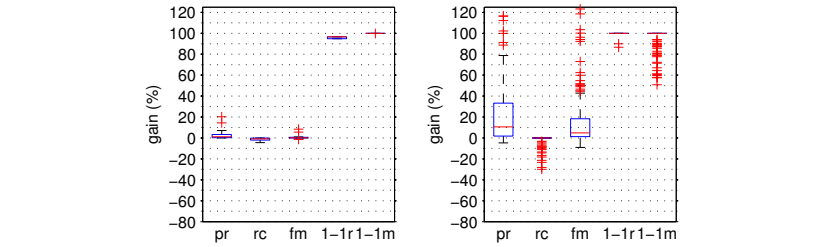


(a) Anatomy. (b) Conference.

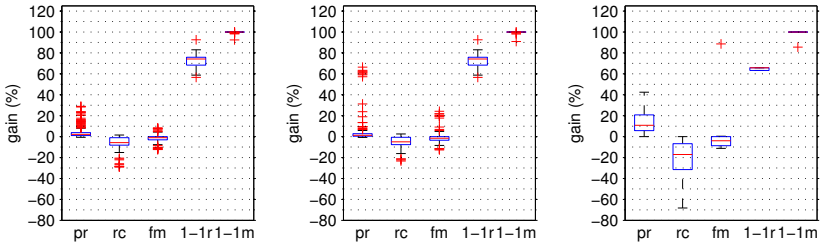


(c) LargeBio (Big). (d) LargeBio (Small). (e) Library.

Figure 5.10: Equivalence repair effect on precision, recall and f-measure.



(a) Anatomy. (b) Conference.



(c) LargeBio (Big). (d) LargeBio (Small). (e) Library.

Figure 5.11: 1-1 filtering and equivalence repair effect on precision, recall and f-measure.

5.5.6 Runtime Analysis

In this section we analyze the percentage of time spent by our prototype for each of the subtasks required to compute a diagnosis for a given alignment. For this experiment the size of the heap reserved to the JVM was 24 GB, in order to reduce the influence of the Garbage Collector execution.

As expected, Figure 5.12 shows that, independently from the input ontologies and alignment size, computing a (nonconservative) diagnosis using *ASP* on the extracted 1-1 alignment (rightmost bargraph), is faster than directly computing a (nonconservative) diagnosis with the same program (leftmost bargraph), and that it scales better, even for extremely massive ontologies (as for *largebio-big* track). The number of alignments contributing to the result is reported at the bottom of each graph.

Note that in the graphs related to the computation of diagnosis on 1-1 alignments, diagnosis time includes also the time spent for filtering the original alignment. In addition, edges loading time always includes the classification of the input ontologies.

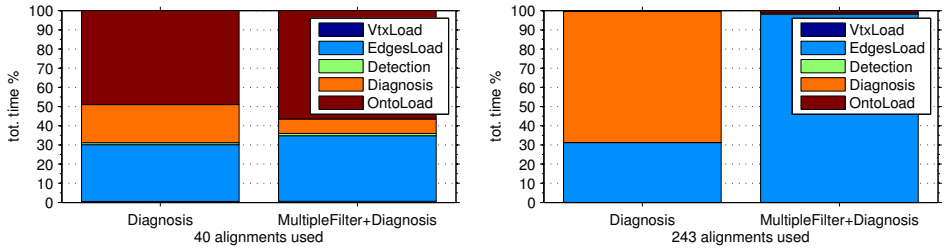
In general, the alignments requiring more time are those with a low percentage of 1-1 mappings. This clearly increases the computation time of the *ASP* solver, given that multiple occurrences, especially with similar or same weights, prevent an effective pruning of the search space, that consists of a high number of (almost) equivalent solutions.

It is not surprising that the percentage of time spent for the diagnosis computation is higher for *largebio-small* than for *largebio-big* track. For *largebio-big* track, indeed, the massive size of the input ontologies (see Table 5.5) poses some limits to the scalability of the matching systems, that typically compute smaller alignments w.r.t. the same ontologies of the *largebio-small* track. This results, on average, in a higher number and size of the problematic SCCs for *largebio-small* track than for *largebio-big* track. In addition, due to the higher size of the input ontologies for the *largebio-big* track, the loading times are a relevant fraction of the total runtime, thus reducing, in percentage, that of the diagnosis computation.

For what concerns *library* track, the diagnosis computation time dominates in both cases. This is motivated by the extremely high size and number of problematic SCCs. In addition, XMapGen, Hertuda and MaasMatch matchers presented huge SCCs, for which also the time required by multiple-occurrences filtering was not negligible. Consider also that classification time for the input ontologies is reduced (less than 250 ms), despite the ontology size.

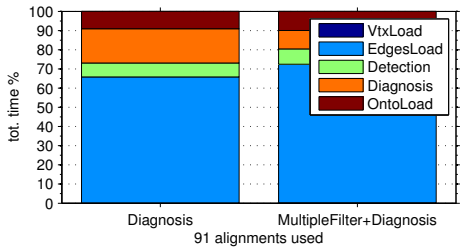
In conclusion, as a general trend, the percentage of 1-1 mappings in an alignment seems to have a higher impact on the percentage spent on diagnosis computation than the size of the input ontologies and that of the alignment itself.

The analysis provided in this section, detailed for each matcher in isolation, is available in [Sol15] Section 1.3.

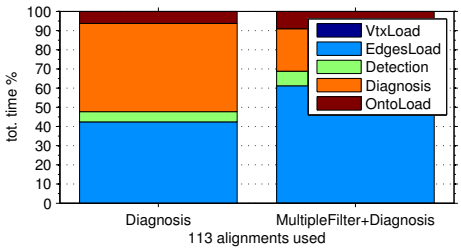


(a) Anatomy.

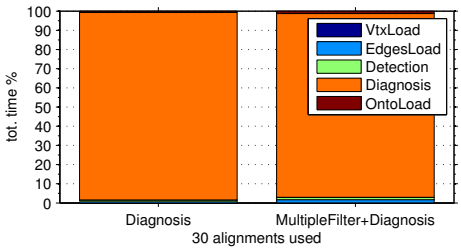
(b) Conference.



(c) LargeBio (Big).



(d) LargeBio (Small).



(e) Library.

Figure 5.12: Equivalence repair subtasks percentage of total diagnosis computation time.

5.5.7 1-1 Alignment Extraction

In this section we compare the runtime and quality of the heuristic presented in Algorithm 15 with the optimal filtering computed using the *ASP* program shown in Listing 5.5 (both introduced in Section 5.4.3). The quality of the filtering, in accordance to the principle of minimal change, is computed as the total sum of the filtered (*i.e.*, removed) mappings.

The experimental results of Figure 5.13 show that the quality of the 1-1 alignment (*i.e.*, total weight of removed mappings) extracted by means of the heuristic is generally close to the optimal one computed using *ASP*. In particular, it is also possible, for the heuristic, to compute a solution with lower total weight than *ASP*, when the solver exceeds the timeout value (see, for instance, the results for $|\mathcal{M}|$ equal to $\sim$76 and $\sim$120). In terms of scalability, the runtime of the two solutions are comparable for small alingments. For alignments composed by more than $\sim$60

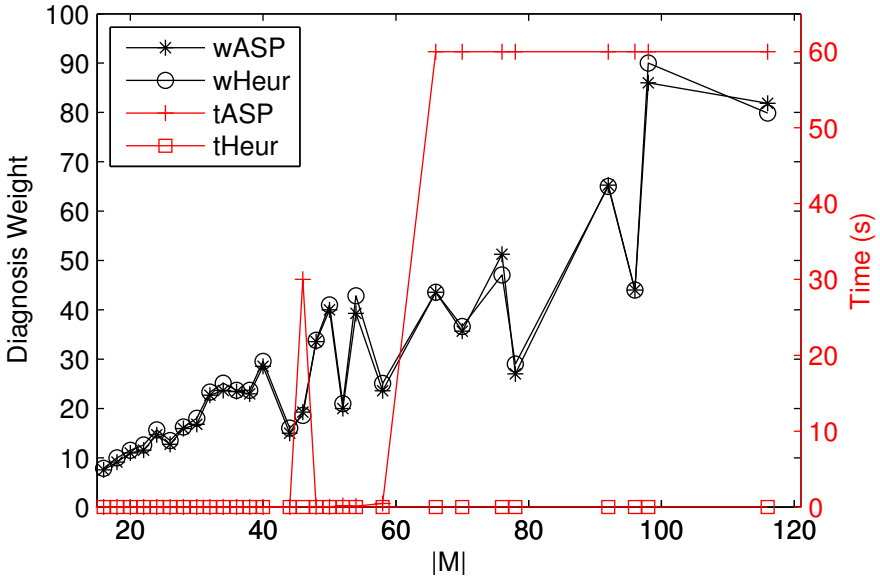


Figure 5.13: Runtime and quality comparison between an optimal (*ASP*-based) 1-1 filtering algorithm and an heuristic algorithm for increasing alignment size.

mappings, the runtime required by the *ASP*-based program rapidly increases, but due to the imposed timeout we do not see the real increase. Instead, a more stable computation time is exhibited by the heuristic approach. Note that the temporal complexity of our heuristic mainly depends on the alignment size (as discussed in Proposition C.6, Section C.2). *ASP* solver, instead, applies different optimization strategies that exploit the *ASP* program structure (facts and other rules), therefore the practical temporal complexity does not uniquely depend on alignment size.

5.5.8 UMLS Alignments Analysis

In this section, similarly to Section 5.5.3, we analyze some statistics of the debugging of the 2012-2014 *UMLS* alignment, employed as (silver-standard) reference alignments in the *largebio* track of *OAEI*. In addition to few corrections, the 2013 version of the alignments presents fine grained confidence values, where the 2012 version used no confidence values at all (they are set equal to 1, by default). From Table 5.8 we can see that the data of interest for our analysis exactly coincides between versions 2012 and 2013. For this reason, if not necessary, we do not identify an alignment by the version, but only using the input ontologies it refers to. Instead, version 2014[13] differs from the previous ones and exhibits, in general, a higher percentage of 1-1 mappings, and less violations (*i.e.*, less problematic SCCs and less problematic mappings).

[13]Version 2014 coincides with version 2013, at the exception of the marking of conflictive mappings w.r.t. consistency principle, with an undefined semantic relation. These mappings were not considered in the present evaluation.

	% D/PM	% PM/M	% D/M	% 1-1M	$\|\mathcal{M}\|$	% VtxSCC/Vtx	probSCC	ASP (s)	% OptDiag
fmanci-2012	25.65	31.78	8.15	68.32	6,048	0.91	414	4.85	100
fmanci-2013	25.75	31.78	8.18	68.32	6,048	0.91	414	4.52	100
fmanci-2014	25.15	25.09	6.31	75.02	5,372	0.65	303	3.23	100
fmasnomed-2012	25.94	25.38	6.58	74.67	18,016	1.58	995	13.28	100
fmasnomed-2013	26.01	25.38	6.6	74.67	18,016	1.58	995	14.05	100
fmasnomed-2014	25.9	25.27	6.55	74.78	12,052	1.05	666	8.85	100
snomednci-2012	27.76	43.48	12.07	56.67	37,688	5.77	3,055	35.73	100
snomednci-2013	28.07	43.48	12.21	56.67	37,688	5.77	3,055	36.51	100
snomednci-2014	27.88	38.98	10.87	61.17	34,420	4.83	2,613	32.76	100

Table 5.8: Analysis of the *UMLS* alignments (2012-2014 versions).

The degree of problematicity is high for all the considered alignments, as well as the number of problematic SCCs. Concerning the behaviour of our violation repair algorithm, the diagnoses for the 2013 and 2014 versions are more accurate than those for the 2012 version, due to the finer-grained confidence values of the newer versions. We discuss relevant concepts by means of examples extracted from the output of the alignment between *FMA* and *NCI* ontologies (the first three rows of Table 5.8). For sake of conciseness, we do not discuss the whole set of problematic SCCs that has been detected in *UMLS* 2012-2014 alignments.[14] In the examples, an arc (A, B, c) of the graph representation, with A a concept of the first input ontology, and B a concept belonging to the second input ontology, is represented as 1_A -> (c) 2_B. We refer to the diagnosis for a problematic SCC, as local diagnosis. A diagnosis for a SCC originating from the $201X$ version of the *UMLS* alignments, is denoted as Δ_{1X}.

In Example 5.20, from the biological domain knowledge, we know that the *islets of Langerhans* are the regions of the pancreas that contain its endocrine cells. An endocrine cell, called *islet cell* in this context, is a cell responsible for hormone production. Therefore, we may encode our domain knowledge with the following axioms, $Pancreatic\_islet \equiv Islet\_of\_Langerhans$ and $Endocrine\_pancreas \equiv Set\_of\_pancreatic\_islets$.

It is evident that the problematic SCC does not match biological background knowledge. In particular, we know that it is not a correct logical consequence that *islet cells* are equivalent to any of the other classes, and that *islet of Langerhans* and *pancreatic islet* are not equivalent to *endocrine Pancreas*. This confirms that the SCC detected as problematic, indeed it contains some incorrect logical entailments.

Comparing diagnosis Δ_{12} with Δ_{13} (identical to Δ_{14}), we note that the quality of the second one is higher. In particular, both diagnoses do not remove all the problamtic entailments, given that part of them relates to conservativity principle violations w.r.t. subsumption, that is out of scope for the present chapter.
Δ_{12}, however, wrongly removes two axioms out of six (that is, $1\_Endocrine\_pancreas \sqsubseteq 2\_Endocrine\_Pancreas$ and $1\_Pancreatic\_islet \sqsubseteq 2\_Islet\_of\_Langerhans$), while Δ_{13} wrongly removes only axiom $1\_Set\_of\_pancreatic\_islets \sqsubseteq 2\_Endocrine\_Pancreas$, out of

[14] Available at `ftp://ftp.disi.unige.it/person/SolimandoA/umlsFull.zip`

six. Both diagnoses are minimal, but Δ_{13} achieves better accuracy by reflecting more fine-grained confidence values for the mappings in the original alignment.

Example 5.20. FMA->NCI (2012):

```
Local Diagnosis: [
1_Endocrine_pancreas -> (1.0) 2_Endocrine_Pancreas,
1_Pancreatic_islet -> (1.0) 2_Endocrine_Pancreas,
1_Islet_cell -> (1.0) 2_Endocrine_Pancreas,
2_Islet_of_Langerhans -> (1.0) 1_Set_of_pancreatic_islets,
1_Pancreatic_islet -> (1.0) 2_Islet_of_Langerhans,
1_Endocrine_pancreas -> (1.0) 2_Islet_of_Langerhans
]

SCC: [2_Islet_of_Langerhans, 1_Pancreatic_islet, 1_Islet_cell,
1_Set_of_pancreatic_islets, 2_Endocrine_Pancreas, 1_Endocrine_pancreas]
Mappings: [2_Islet_of_Langerhans -> (1.0) 1_Pancreatic_islet,
 1_Endocrine_pancreas -> (1.0) 2_Endocrine_Pancreas,
 2_Endocrine_Pancreas -> (1.0) 1_Endocrine_pancreas,
 1_Islet_cell -> (1.0) 2_Islet_of_Langerhans,
 1_Pancreatic_islet -> (1.0) 2_Endocrine_Pancreas,
 1_Islet_cell -> (1.0) 2_Endocrine_Pancreas,
 1_Set_of_pancreatic_islets -> (1.0) 2_Islet_of_Langerhans,
 2_Islet_of_Langerhans -> (1.0) 1_Endocrine_pancreas,
 2_Endocrine_Pancreas -> (1.0) 1_Pancreatic_islet,
 2_Islet_of_Langerhans -> (1.0) 1_Set_of_pancreatic_islets,
 2_Islet_of_Langerhans -> (1.0) 1_Islet_cell,
 1_Set_of_pancreatic_islets -> (1.0) 2_Endocrine_Pancreas,
 1_Pancreatic_islet -> (1.0) 2_Islet_of_Langerhans,
 1_Endocrine_pancreas -> (1.0) 2_Islet_of_Langerhans,
 2_Endocrine_Pancreas -> (1.0) 1_Set_of_pancreatic_islets,
 2_Endocrine_Pancreas -> (1.0) 1_Islet_cell]
Edges: []


FMA->NCI (2013/2014):

Local Diagnosis: [
1_Islet_cell -> (0.48) 2_Islet_of_Langerhans,
1_Pancreatic_islet -> (0.11) 2_Endocrine_Pancreas,
1_Set_of_pancreatic_islets -> (0.5) 2_Islet_of_Langerhans,
1_Islet_cell -> (0.0) 2_Endocrine_Pancreas,
2_Islet_of_Langerhans -> (0.5) 1_Endocrine_pancreas,
1_Set_of_pancreatic_islets -> (0.22) 2_Endocrine_Pancreas
]

SCC: [2_Islet_of_Langerhans, 1_Pancreatic_islet, 1_Islet_cell,
1_Set_of_pancreatic_islets, 2_Endocrine_Pancreas, 1_Endocrine_pancreas]
Mappings: [2_Islet_of_Langerhans -> (0.56) 1_Pancreatic_islet,
 1_Endocrine_pancreas -> (0.5) 2_Endocrine_Pancreas,
 2_Endocrine_Pancreas -> (0.5) 1_Endocrine_pancreas,
 1_Islet_cell -> (0.48) 2_Islet_of_Langerhans,
 1_Pancreatic_islet -> (0.11) 2_Endocrine_Pancreas,
 1_Islet_cell -> (0.0) 2_Endocrine_Pancreas,
 1_Set_of_pancreatic_islets -> (0.5) 2_Islet_of_Langerhans,
 2_Islet_of_Langerhans -> (0.5) 1_Endocrine_pancreas,
 2_Endocrine_Pancreas -> (0.11) 1_Pancreatic_islet,
 2_Islet_of_Langerhans -> (0.5) 1_Set_of_pancreatic_islets,
 2_Islet_of_Langerhans -> (0.48) 1_Islet_cell,
 1_Set_of_pancreatic_islets -> (0.22) 2_Endocrine_Pancreas,
 1_Pancreatic_islet -> (0.56) 2_Islet_of_Langerhans,
```

```
 1_Endocrine_pancreas -> (0.5) 2_Islet_of_Langerhans,
 2_Endocrine_Pancreas -> (0.22) 1_Set_of_pancreatic_islets,
 2_Endocrine_Pancreas -> (0.0) 1_Islet_cell]
Edges: []
```

◊

Examples 5.21-5.23 presents three problematic SCCs, belonging to both versions of *UMLS*. In these three cases, the role played by a fine-grained choice of confidence values, clearly emerges. Consider, for instance, the SCC of Example 5.21. Despite both Δ_{12} and Δ_{13} (identical to Δ_{14}) are correct minimal diagnoses, only Δ_{13} is semantically accurate, given that both concepts for auditory ossicle are equivalent (*i.e.*, $1\_Auditory\_Ossicle \sqsubseteq 2\_Auditory\_Ossicle$), but an auditory ossicle could be also seen as the singleton of auditory ossicle composed exclusively by itself (*i.e.*, axiom $2\_Auditory\_Ossicle \sqsubseteq 2\_Set\_of\_auditory\_ossicles$ holds, but the two concepts are not equivalent).

Example 5.21. An example of violation of the conservativity principle in the *UMLS* alignments between *FMA* and *NCI* ontologies, version 2012-2014. One of the effects of this violation is the entailment, in the aligned ontology, of the incorrect axiom $1\_Auditory\_Ossicle \equiv 2\_Set\_of\_auditory\_ossicles$.

```
FMA->NCI (2012):

Local Diagnosis: [2_Auditory_Ossicle -> (1.0) 1_Auditory_ossicle]

SCC: [2_Auditory_Ossicle, 1_Set_of_auditory_ossicles, 1_Auditory_ossicle]
Mappings: [2_Auditory_Ossicle -> (1.0) 1_Auditory_ossicle,
 1_Set_of_auditory_ossicles -> (1.0) 2_Auditory_Ossicle,
 2_Auditory_Ossicle -> (1.0) 1_Set_of_auditory_ossicles,
 1_Auditory_ossicle -> (1.0) 2_Auditory_Ossicle]
Edges: []


FMA->NCI (2013/2014):

Local Diagnosis: [1_Set_of_auditory_ossicles -> (0.49) 2_Auditory_Ossicle]

SCC: [2_Auditory_Ossicle, 1_Set_of_auditory_ossicles, 1_Auditory_ossicle]
Mappings: [2_Auditory_Ossicle -> (0.67) 1_Auditory_ossicle,
 1_Set_of_auditory_ossicles -> (0.49) 2_Auditory_Ossicle,
 2_Auditory_Ossicle -> (0.49) 1_Set_of_auditory_ossicles,
 1_Auditory_ossicle -> (0.67) 2_Auditory_Ossicle]
Edges: []
```

◊

For what concerns Example 5.22, *NCI* ontology contains axiom $2\_Interleukin\text{-}3 \sqsubseteq 2\_Hematopoietic\_Growth\_Factor$, and we therefore know that *Interleukin-3* is a *hematopoietic growth factor*. From the domain knowledge, in addition, we know that they

are not equivalent. The problematic SCC shown in Example 5.22, instead, asserts the equivalence between these two concepts. Again, the more fine-grained confidence values of the 2013/2014 version of the *UMLS* alignments enable the computation of a semantically correct diagnosis, while the same does not hold for Δ_{12}.

Example 5.22. An example of violation of the conservativity principle in the *UMLS* alignment between *FMA* and *NCI* ontologies, versions 2012-2014. One of the effects of this violation is the entailment, in the aligned ontology, of the incorrect axiom $2\_Interleukin\text{-}3 \equiv 2\_Hematopoietic\_Growth\_Factor$.

```
FMA->NCI (2012):

Local Diagnosis: [1_Interleukin_3 -> (1.0) 2_Interleukin-3]

SCC: [2_Hematopoietic_Growth_Factor, 2_Interleukin-3, 1_Interleukin_3]
Mappings: [1_Interleukin_3 -> (1.0) 2_Hematopoietic_Growth_Factor,
 2_Interleukin-3 -> (1.0) 1_Interleukin_3,
 1_Interleukin_3 -> (1.0) 2_Interleukin-3,
 2_Hematopoietic_Growth_Factor -> (1.0) 1_Interleukin_3]
Edges: [2_Interleukin-3 -> (1.0) 2_Hematopoietic_Growth_Factor]


FMA->NCI (2013/2014):

Local Diagnosis: [2_Hematopoietic_Growth_Factor -> (0.11) 1_Interleukin_3]

SCC: [2_Hematopoietic_Growth_Factor, 2_Interleukin-3, 1_Interleukin_3]
Mappings: [1_Interleukin_3 -> (0.11) 2_Hematopoietic_Growth_Factor,
 2_Interleukin-3 -> (0.68) 1_Interleukin_3,
 1_Interleukin_3 -> (0.68) 2_Interleukin-3,
 2_Hematopoietic_Growth_Factor -> (0.11) 1_Interleukin_3]
Edges: [2_Interleukin-3 -> (1.0) 2_Hematopoietic_Growth_Factor]
```

$\Diamond$

From the biological domain,[15] we know that "there are three types of myofilaments, that is thick, thin, and elastic filaments". In particular, we know that "thin filaments consist primarily of the protein actin". Therefore, we can state that $Myofilament \sqsubseteq Actin\ Filament$, but they are not equivalent, and that $Thin\ filaments \equiv Actin\ filaments$.

From the background knowledge we therefore know that the problematic SCC of Example 5.23 is not correct, given that it entails $Actin\ filament \sqsubseteq Myofilament$ for both input ontologies. Again, Δ_{13} (identical to Δ_{14}) is semantically correct, while Δ_{12} it is not.

Example 5.23. An example of violation of the conservativity principle in the *UMLS* alignments between *FMA* and *NCI* ontologies, versions 2012-2014. One of the effects of this violation is the entailment, in the aligned ontology, of the incorrect axioms $1\_Actin\_filament \equiv 1\_Myofilament$ and $2\_Actin\_Filament \equiv 2\_Myofilament$.

[15] The information has been extracted from `http://en.wikipedia.org/wiki/Actin_filaments`

```
FMA->NCI (2012):

Local Diagnosis: [2_Myofilament -> (1.0) 1_Myofilament,
1_Actin_filament -> (1.0) 2_Actin_Filament]

SCC: [2_Actin_Filament, 2_Myofilament, 1_Myofilament,
1_Actin_filament]
Mappings: [2_Actin_Filament -> (1.0) 1_Actin_filament,
 2_Actin_Filament -> (1.0) 1_Myofilament,
 2_Myofilament -> (1.0) 1_Actin_filament,
 1_Myofilament -> (1.0) 2_Myofilament,
 1_Myofilament -> (1.0) 2_Actin_Filament,
 1_Actin_filament -> (1.0) 2_Myofilament,
 2_Myofilament -> (1.0) 1_Myofilament,
 1_Actin_filament -> (1.0) 2_Actin_Filament]
Edges: []


FMA->NCI (2013/2014):

Local Diagnosis: [2_Actin_Filament -> (0.54) 1_Myofilament,
1_Actin_filament -> (0.45) 2_Myofilament]

SCC: [2_Actin_Filament, 2_Myofilament, 1_Myofilament,
1_Actin_filament]
Mappings: [2_Actin_Filament -> (0.69) 1_Actin_filament,
 2_Actin_Filament -> (0.54) 1_Myofilament,
 2_Myofilament -> (0.45) 1_Actin_filament,
 1_Myofilament -> (0.6) 2_Myofilament,
 1_Myofilament -> (0.54) 2_Actin_Filament,
 1_Actin_filament -> (0.45) 2_Myofilament,
 2_Myofilament -> (0.6) 1_Myofilament,
 1_Actin_filament -> (0.69) 2_Actin_Filament]
Edges: []
```

◊

However, in all the Examples 5.21-5.23, the minimal diagnosis is not unique. This is due to the fact that *UMLS* uses almost only equivalence semantic relation, and therefore both arcs originating from each single mapping of this kind, present the same confidence value. Therefore, whenever one of the two belongs to a minimal diagnosis, another minimal diagnosis having this element replaced by the opposite arc exists. In cases of multiple minimal diagnoses, the problem is inherently ambigous for an automatic approach, and a manual revision could be suggested. In all the three cases, the other minimal diagnosis is not semantically correct w.r.t. the background knowledge.

However, as discussed in Section 5.2, violations of the conservativity principle (both w.r.t. equivalence and subsumption) may be false positive. Consider, for instance, Example 5.24. In this case we know that *forebrain* is a synonym of *prosencephalon*, and therefore the two concepts are equivalent. Despite this, *NCI* ontology does not explicitly state it through an axiom. For this reason, the problematic SCC has to be considered a false positive. In such cases, our automatic approach computes a diagnosis and partially removes correct knowledge. In these cases, the adequate support for a manual revision could enable the inspection of such violations by a

manual curator, that could eventually integrate the input ontology with the missing equivalence axiom (as suggested in the method by Lambrix *et al.* [LWKDI13, LDI13]). This on one hand would solve the violation, and on the other could be a suggestion for integrating the knowledge of the input ontologies, that could have been simply missed by the modeler (this, for instance, may happen quite easily for biomedical ontologies, considering their massive size).

Example 5.24. An example of violation of the conservativity principle in the *UMLS* alignments between *FMA* and *NCI* ontologies, versions 2012-2014. In this case, one of the effects of the violation is the entailment of the axiom $2\_Fore - Brain \equiv 2\_Prosencephalon$, that is correct, even if not explicitly stated in the *NCI* ontology.

```
FMA->NCI (2012):

Local Diagnosis: [1_Forebrain -> (1.0) 2_Prosencephalon]

SCC: [2_Fore-Brain, 2_Prosencephalon, 1_Forebrain]
Mappings: [2_Fore-Brain -> (1.0) 1_Forebrain,
 1_Forebrain -> (1.0) 2_Fore-Brain,
 2_Prosencephalon -> (1.0) 1_Forebrain,
 1_Forebrain -> (1.0) 2_Prosencephalon]
Edges: []


FMA->NCI (2013/2014):

Local Diagnosis: [1_Forebrain -> (0.55) 2_Prosencephalon]

SCC: [2_Fore-Brain, 2_Prosencephalon, 1_Forebrain]
Mappings: [2_Fore-Brain -> (0.55) 1_Forebrain,
 1_Forebrain -> (0.55) 2_Fore-Brain,
 2_Prosencephalon -> (0.55) 1_Forebrain,
 1_Forebrain -> (0.55) 2_Prosencephalon]
Edges: []
```

◊

5.5.9 Support for Manual Alignment Debugging

In this section we analyze the features provided by our tool in order to support manual alignments debugging by means of a step-by-step example.

The first step consists in loading an alignment and the associated pair of input ontologies. At this point it is possible to compute (and list) the identified problematic SCCs, if any.

Then, an intermediate step allows the user to visualize and repair the problematic SCCs. It is possible to select any problematic SCC from a list in the leftmost part of the program (an example is given in Figure 5.14). In addition to the selected SCC, the program highlights the problematic SCCs for which a diagnosis has been computed. This feature helps to keep track of the SCCs that still need to be analyzed (an example is provided in Figure 5.14).

When a problematic SCC is selected, the GUI shows a graphical representation of it (in the central panel) within the list of its mappings. An example of the mappings list is given in Figure 5.14 (the rightmost one).

For any selected problematic SCC it is possible to:

- extract (from its set of mappings) a maximal set of 1-1 mappings w.r.t. its total weight, in accordance to the principle of minimal change,
- compute a minimal diagnosis, with an optional selection of a subset of the mappings that cannot be removed, called *sealed* mappings.

Figure 5.15 shows the resulting 1-1 mappings for the problematic SCC depicted in Figure 5.14.

If the diagnosis search space for the current combination of SCC and selection sealed mappings is empty, the program informs the user that the problem is unsatisfiable. Otherwise, the minimal diagnosis is computed, and removed mappings are highlighted in the mapping list. Figure 5.16 shows a selection of sealed mappings that leads to an unsatisfiable instance of the diagnosis computation problem.

The last step occurs when the user considers all the problems solved (*i.e.*, all the problematic SCCs that are not false positives are repaired). This step allows the user to save the alignment in the *Alignment API* 4.7 format (RDF-based, supported also by *OAEI*), that can be loaded and processed by any *Alignment API* compliant program.

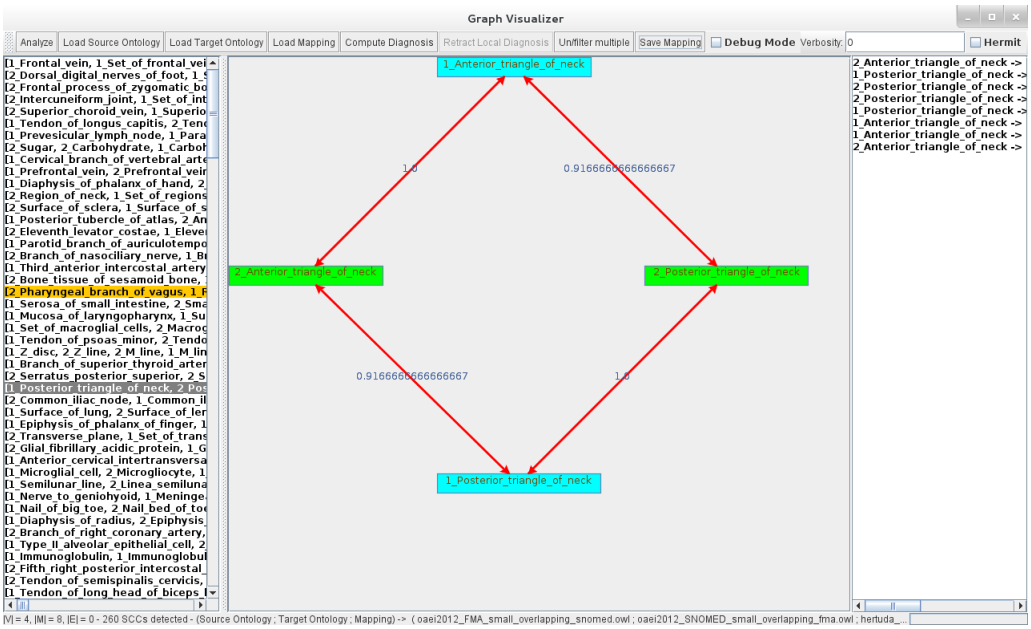


Figure 5.14: Selection of an unrepaired problematic SCC.

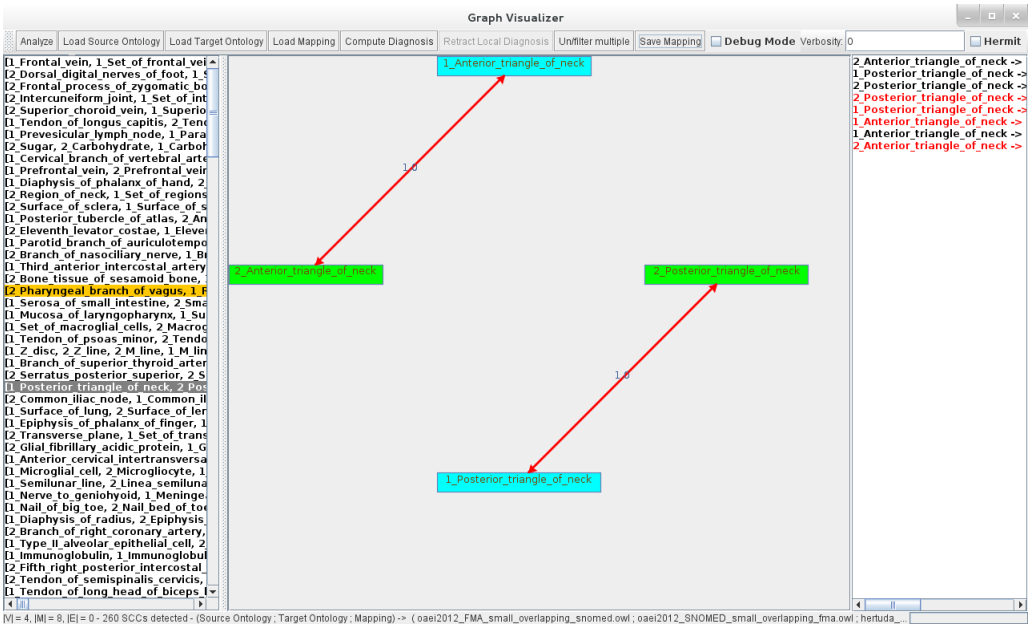


Figure 5.15: 1-1 mappings extraction for a problematic SCC.

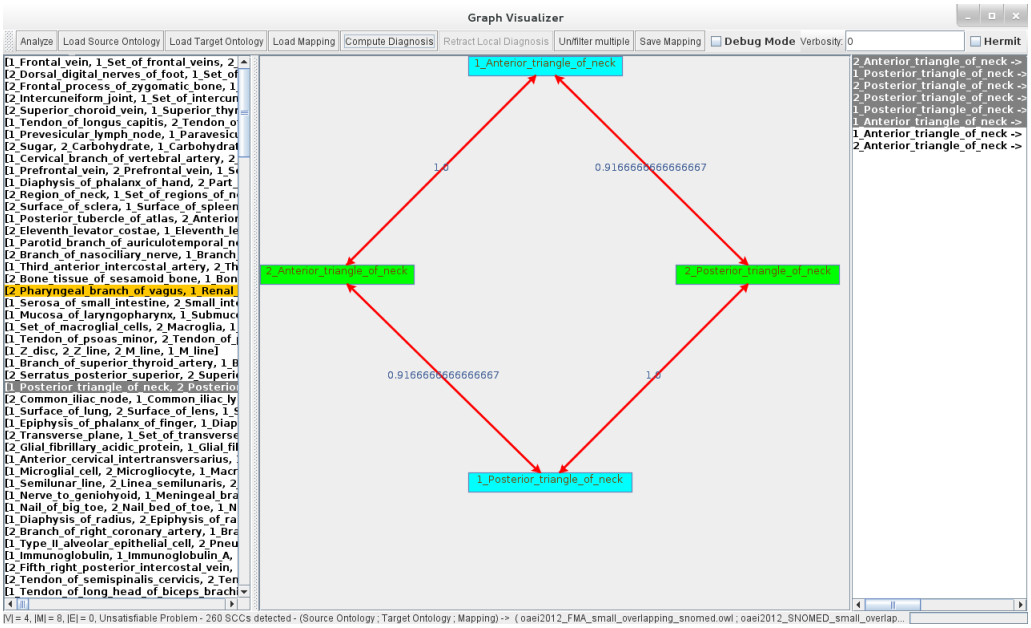


Figure 5.16: Selection of sealed mappings that can not be removed by the diagnosis for the selected problematic SCC.

Chapter 6

Subsumption Conservativity Principle Violations

6.1 Introduction

Starting from the formal definitions of the different kinds of conservativity principle violations (Section 4.3.6), it is evident that the techniques proposed for detecting and correcting equivalence violations are not applicable for subsumption violations.

The motivation is that subsumption violations, and basic violations as well, cannot be characterized in terms of strongly connected components or cycles, as equivalence violations, and for this reason require specific detection and repair techniques (a concrete example is provided in Section 6.3).

Specifically, in this chapter we aim at introducing a detection technique covering also subsumption violations, based on efficient indexing of the classification DAG, and we also aim at reducing conservativity principle problem to a consistency repair problem.

The present chapter, mainly focuses on basic violations, and for this reason we generally refer to these violations simply as violations, but when other notions are used, they are explicitly mentioned.

The structure of the chapter sees the related work covered by Section 6.2, and the problem stated in Section 6.3. The proposed methods and algorithms are then discussed in Section 6.4, and experimentally evaluated in Section 6.5.

6.2 Related Work

As already discussed in Chapters 4 and 5, conservativity principle, although indirectly, has been actively studied in the literature. This section provides an overview of the related work for the assumption of disjointness, tightly related to the technique presented in this chapter.

The assumption of disjointness (already presented in Section 4.3.6) has been originally introduced by Schlobach [Sch05] to enhance the repair of ontologies that were underspecified in terms of disjointness axioms.

In [MVS08], a similar assumption is followed in the context of ontology mappings repair, where the authors restricted the number of disjointness axioms by using learning techniques [VVSH07, FV11]. These techniques, however, typically require a manually created training set.

In [FR12] the authors present an interactive system to guide the expert user in the manual enrichment of the ontologies with negative constraints, including disjointness axioms.

Our proposal, as [VVSH07, MVS08, FV11, FR12], aims at adding a small set of disjointness axioms, since adding all possible disjointness may be unfeasible for large ontologies. However, our method does not require manual intervention. Furthermore, to address the scalability problem when dealing with large ontologies and mapping sets, our method relies on the propositional projection of the input ontologies.

6.3 Problem Statement

In this section, we first show (in Section 6.3.1) the problems led by the violation of the conservativity principle when integrating ontologies via mappings in a real-world scenario. To this end, we consider as motivating example a use case based on the *Optique* project application domain.[1] Section 6.3.2 then introduces the basic notations and definitions used in the remaining sections of the chapter.

6.3.1 Motivating Example

Optique aims at facilitating scalable end-user access to big data in the oil and gas industry. Optique advocates for an OBDA approach [KJRZ$^+$13, GHJR$^+$15] so that end-users formulate queries using the vocabulary of a domain ontology instead of composing queries directly against the database. Ontology-based queries (*e.g.*, SPARQL queries) are then automatically rewritten to SQL and executed over the database. Ontology entities are linked to the database through

[1] Optique project has been already briefly introduced in Section 4.7.2.

	Ontology $\mathcal{O}_1$		Ontology $\mathcal{O}_2$
α_1	$WellBore \sqsubseteq \exists belongsTo.Well$	β_1	$Exploration\_well \sqsubseteq Well$
α_2	$WellBore \sqsubseteq \exists hasOperator.Operator$	β_2	$Explor\_borehole \sqsubseteq Borehole$
α_3	$WellBore \sqsubseteq \exists locatedIn.Field$	β_3	$Appraisal\_exp\_borehole \sqsubseteq Explor\_borehole$
α_4	$AppraisalWellBore \sqsubseteq WellBore$	β_4	$Appraisal\_well \sqsubseteq Well$
α_5	$ExplorationWellBore \sqsubseteq WellBore$	β_5	$Field \sqsubseteq \exists hasFieldOperator.Field\_operator$
α_6	$Operator \sqsubseteq Owner$	β_6	$Field\_operator \sqcap Owner \sqsubseteq Field\_owner$
α_7	$Operator \sqsubseteq Company$	β_7	$Company \sqsubseteq Field\_operator$
α_8	$Field \sqsubseteq \exists hasOperator.Company$	β_8	$Field\_owner \sqsubseteq Owner$
α_9	$Field \sqsubseteq \exists hasOwner.Owner$	β_9	$Borehole \sqsubseteq Continuant \sqcup Occurrent$

Table 6.1: Simplified fragments of two ontologies in the oil and gas domain.

Mappings $\mathcal{M}$				
id	$\mathbf{e_1}$	$\mathbf{e_2}$	**n**	$\boldsymbol{\rho}$
m_1	$\mathcal{O}_1$:$Well$	$\mathcal{O}_2$:$Well$	0.9	$\equiv$
m_2	$\mathcal{O}_1$:$WellBore$	$\mathcal{O}_2$:$Borehole$	0.7	$\equiv$
m_3	$\mathcal{O}_1$:$ExplorationWellBore$	$\mathcal{O}_2$:$Exploration\_well$	0.6	$\sqsubseteq$
m_4	$\mathcal{O}_1$:$ExplorationWellBore$	$\mathcal{O}_2$:$Explor\_borehole$	0.8	$\equiv$
m_5	$\mathcal{O}_1$:$AppraisalWellBore$	$\mathcal{O}_2$:$Appraisal\_exp\_borehole$	0.7	$\equiv$
m_6	$\mathcal{O}_1$:$Field$	$\mathcal{O}_2$:$Field$	0.9	$\equiv$
m_7	$\mathcal{O}_1$:$Operator$	$\mathcal{O}_2$:$Field\_operator$	0.7	$\sqsupseteq$
m_8	$\mathcal{O}_1$:$Company$	$\mathcal{O}_2$:$Company$	0.9	$\equiv$
m_9	$\mathcal{O}_1$:$hasOperator$	$\mathcal{O}_2$:$hasFieldOperator$	0.6	$\equiv$
m_{10}	$\mathcal{O}_1$:$Owner$	$\mathcal{O}_2$:$Owner$	0.9	$\equiv$

Table 6.2: Ontology mappings for the vocabulary in $\mathcal{O}_1$ and $\mathcal{O}_2$.

ontology-to-schema mappings. Ontology-to-schema mappings are, however, out of the scope of the present thesis, and we therefore only concentrate on ontology-to-ontology mappings, that will be referred to simply as mappings when no confusion arises.

Table 6.1 shows two simplified fragments of ontologies that are currently being used in the context of Optique. The ontology $\mathcal{O}_1$ has been directly bootstrapped from a relational database in Optique, and it is linked to the data through direct ontology-to-database mappings. The ontology $\mathcal{O}_2$, instead, is a domain ontology, based on the NPD FactPages, preferred by Optique end-users to feed the visual query formulation interface [SSG+13].[2]

The integration via ontology matching of $\mathcal{O}_1$ and $\mathcal{O}_2$ is required since the vocabulary in $\mathcal{O}_2$ is used to formulate queries, but only the vocabulary of $\mathcal{O}_1$ is connected to the database. Consider the set of mappings $\mathcal{M}$ in Table 6.2 between $\mathcal{O}_1$ and $\mathcal{O}_2$ generated by an off-the-shelf ontology matching system. As described in Definition 4.17 (Section 4.3), mappings are represented as 4-tuples; for example the mapping m_2 suggests an equivalence relationship between the entities $\mathcal{O}_1$:$WellBore$ and $\mathcal{O}_2$:$Borehole$, with confidence 0.7.

The integrated ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$, however, violates the conservativity principle, according to Definition 4.26, and introduces undesired entailments (see Table 6.3). In Figure 6.1

[2]Optique uses OWL 2 QL ontologies for query rewriting, while the query formulation may be based on much richer OWL 2 ontologies. The axioms that fall outside the OWL 2 QL profile are either approximated or not considered for the rewriting.

σ	Entailment:	follows from:	basicViol?	eqViol?
σ_1	$\mathcal{O}_2$:$Explor\_borehole \sqsubseteq \mathcal{O}_2$:$Exploration\_well$	m_3, m_4	YES	NO
σ_2	$\mathcal{O}_1$:$AppraisalWellBore \sqsubseteq \mathcal{O}_1$:$ExplorationWellBore$	β_3, m_4, m_5	YES	NO
σ_3	$\mathcal{O}_2$:$Field\_operator \sqsubseteq \mathcal{O}_2$:$Field\_owner$	$\alpha_6, \beta_6, m_7, m_{10}$	YES	NO
σ_4 σ_5	$\mathcal{O}_1$:$Company \equiv \mathcal{O}_1$:$Operator$ $\mathcal{O}_2$:$Field\_operator \equiv \mathcal{O}_2$:$Company$	$\alpha_7, \beta_7, m_7, m_8$	NO	YES
σ_6	$\mathcal{O}_1$:$Company \sqsubseteq \mathcal{O}_1$:$Owner$	σ_4, α_6	YES	NO
σ_7	$\mathcal{O}_2$:$Company \sqsubseteq \mathcal{O}_2$:$Field\_owner$	σ_3, σ_5	YES	NO

Table 6.3: Example of conservativity principle violations.

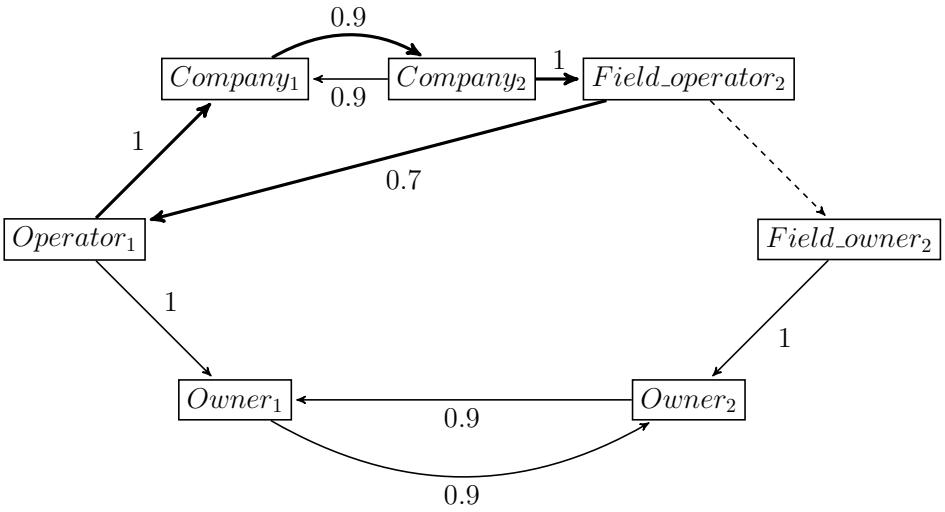


Figure 6.1: Graph representation of the fragment of the aligned ontology of Table 6.1 involved in conservativity violations.

the portion of the graph representation of $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ involved in conservativity violations is shown. Dashed arcs represent inferred axioms, while bold arcs are those involved in equivalence violations. Each non-inferred arc is labeled with its confidence value.

Note that the entailments σ_4 and σ_5 are equivalence violations not belonging to the set of basic (subsumption) violations, since $\mathcal{O}_1$:$Company$ and $\mathcal{O}_1$:$Operator$ (resp. $\mathcal{O}_2$:$Field\_operator$ and $\mathcal{O}_2$:$Company$) are involved in a subsumption relationship in $\mathcal{O}_1$ (resp. $\mathcal{O}_2$). As already discussed in Section 4.3.6, for these entailments we cannot apply the assumption of disjointness, and therefore they cannot be repaired by the approach introduced in this chapter. On the other hand, not all the basic violations of Table 6.3 can be repaired by the algorithm presented in Chapter 5, but only those stemming from equivalence violations. For instance, equivalence violations σ_4 and σ_5 lead to basic violations (namely σ_6 and σ_7), as shown in Figure 6.1, that could be solved by *ASP*-based approach presented in Chapter 5, but the same clearly does not hold for the other basic conservativity violations (σ_1–σ_3).

From this interrelation between the different violation kinds and the impossibility to address all

of them with a single repair algorithm, we can foresee the motivations for the multi-strategy approach that will be discussed in Chapter 7, combining the proposals of Chapter 5 and that of the present one.

We have already discussed in Section 4.7.2 the impact of conservativity violations in query answering tasks. Therefore, we already know that the quality of the mappings in terms of conservativity principle violations directly affects the quality of the query results. Nonetheless, we provide a further example showing how conservativity violations may affect the quality of the results when performing OBDA queries. Example 6.1 provides an instance of query over the vocabulary of $\mathcal{O}_2$ (shown in Table 6.1).

Example 6.1. Consider the following conjunctive query expressed in datalog notation, $CQ(x) \leftarrow \mathcal{O}_2{:}Well(x)$. The query asks for wells and has been formulated from the Optique's query formulation interface, using the vocabulary of $\mathcal{O}_2$. The query is rewritten, according to the ontology axioms and mappings $\beta_1, \beta_4, m_1, m_3, m_4$ in $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$, into the following union of conjunctive queries $UCQ(x) \leftarrow \mathcal{O}_2{:}Well(x) \cup \mathcal{O}_1{:}Well(x) \cup \mathcal{O}_2{:}Exploration\_well(x) \cup \mathcal{O}_2{:}Appraisal\_well(x) \cup \mathcal{O}_1{:}ExplorationWellBore(x) \cup \mathcal{O}_2{:}Explor\_borehole(x)$. Since only the vocabulary of $\mathcal{O}_1$ is linked to the data, the union of conjunctive queries could be simplified as $UCQ(x) \leftarrow Well(x) \cup ExplorationWellBore(x)$, which will clearly lead to non desired results. The original query was only asking for wells, while the rewritten query will also return data about exploration wellbores. $\Diamond$

6.3.2 Preliminaries

In this section we introduce the notations and definitions used in the remainder of the chapter. First, the Horn propositional encoding of OWL 2 ontologies is introduced, then Horn propositional satisfiability and the well-known Dowling-Gallier algorithm implementing it are covered. The preliminaries for the structural indexing and mapping repair used in LogMap (an ontology matching and mapping repair system [JRC11, JRCZH12, JRMCH13]) and reused in our approach, are introduced. Then, we briefly discuss modularization techniques for DL ontologies. Finally, the section concludes with the introduction of the notion of *direct conservativity violations*.

Propositional Horn Encoding. Definition 6.1 provides the definition of the Horn propositional encoding of an OWL 2 ontology.

Definition 6.1. The *Horn propositional encoding of an ontology* $\mathcal{O}$, for which the classification hierarchy has been computed using an OWL 2 reasoner, consists of the following kinds of clauses:

1. $A \rightarrow B$, for each distinct pair of named concepts $A, B \in N_C(\mathcal{O})$ such that $\mathcal{O} \models A \sqsubseteq B$,

2. $A_i \wedge A_j \rightarrow \bot$, with $1 \leq i < j \leq n$, for each disjointness axiom in $\mathcal{O}$, involving named concepts $A_1, \ldots, A_n$ in $N_C(\mathcal{O})$,

3. $A_1 \wedge \ldots \wedge A_n \rightarrow B$, for each distinct set of named concepts $A_1, \ldots, A_n, B$ such that $\mathcal{O} \models A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B$ or $\mathcal{O} \models A_1 \sqcap \ldots \sqcap A_n \equiv B$.

$\triangle$

Note that only clauses of kind 1 benefit from the propedeutic classification step, while for clauses of kinds 2 and 3 we only encode the explictly asserted axioms.

An example of Horn propositional encoding of mappings and OWL 2 axioms is given in Example 6.2.

Example 6.2. Consider the ontologies and mappings in Tables 6.1 and 6.2. The axiom β_6 is encoded as $Field\_operator \wedge Owner \rightarrow Field\_owner$, while the mapping m_2 is translated into rules $\mathcal{O}_1$:$WellBore \rightarrow \mathcal{O}_2$:$Borehole$, and $\mathcal{O}_2$:$Borehole \rightarrow \mathcal{O}_1$:$WellBore$. $\Diamond$

Horn Propositional Satisfiability. The mapping repair process we use exploits the already cited Dowling-Gallier algorithm (D&G) for propositional Horn satisfiability [DG84]. This algorithm relies on a graph encoding of a given Horn propositional formula. Testing the satisfiability of a given formula, reduces to running graph pebbling tests over the corresponding graph. In what follows we provide the necessary definitions and notations from [DG84].

Definition 6.2 introduces the graph encoding of a Horn propositional formula.

Definition 6.2. Given a Horn formula $\mathcal{P} = C_1 \wedge \ldots \wedge C_M$, $G_{\mathcal{P}}$ is a labeled directed graph with $K + 2$ nodes (a node for each propositional letter occurring in $\mathcal{P}$, a node for $\top$, a node for $\bot$), and set of labels $[1 \ldots M]$. It is constructed with i taking values in $[1 \ldots M]$ as follows:

(i) If the *i-th* basic Horn formula in $\mathcal{P}$ is a positive literal Q, there is an arc from $\top$ to Q labeled i.

(ii) If the *i-th* basic Horn formula in $\mathcal{P}$ is of the form $P_1 \wedge \ldots \wedge P_q \rightarrow \bot$, there are q arcs from $P_1, \ldots, P_q$ to $\bot$ labeled i.

(iii) If the *i-th* basic Horn formula in $\mathcal{P}$ is of the form $P_1 \wedge \ldots \wedge P_q \rightarrow Q$, there are q arcs from $P_1, \ldots, P_q$ to Q labeled i.

The graph is called the *graph corresponding* to $\mathcal{P}$. $\triangle$

Definition 6.3 introduces the graph pebbling test used to define Horn satisfiability. G is assumed to be the graph encoding of a given Horn formula $C_1 \wedge \ldots \wedge C_M$.

Definition 6.3. Let $G = (V, A, L)$ be an arc-labeled directed graph. There is a *pebbling* of a node $Q \in V$ from a set of nodes $X \subseteq V$ if either Q belongs to X or, for some label i (corresponding to some basic Horn formula C_i), there are pebblings for $P_1, \ldots, P_q$ from X, where $P_1, \ldots, P_q$ are the sources of all incoming arcs to Q labeled i. $\triangle$

In Section 4.4.1, Definition 4.29, we already introduced the notion of satisfiability of a Horn formula. In Theorem 6.1 the equivalence between Horn satisfiability of a formula and testing graph pebbling for the corresponding graph is stated.

Theorem 6.1. Let $G_{\mathcal{P}} = (V, A, [1 \ldots M])$ be the graph corresponding to a Horn formula $\mathcal{P}$. $\mathcal{P}$ is *satisfiable* iff there is no pebbling of $\bot$ from $\{\top\}$. $\square$

From Theorem 6.1 we have Corollary 6.1, stating the equivalence between derivation in a Horn formula and pebbling in the corresponding graph.

Corollary 6.1. Given a Horn formula $\mathcal{P}$ and a clause $A \rightarrow B$, we have that $\mathcal{P} \models A \rightarrow B$ holds iff a pebbling of B from $\{A\}$ exists. $\square$

An example of a Horn formula, its encoding, and related satisfiability test is provided in Example 6.3 (adapted from [DG84]).

Example 6.3. Let $\mathcal{P}$ be a Horn formula composed by the following clauses:

1. $P_3 \wedge P_4 \rightarrow P_5$,
2. $P_1 \rightarrow P_2$,
3. $P_2 \rightarrow P_1$,
4. $P_3 \rightarrow P_4$,
5. $\top \rightarrow P_3$,
6. $P_1 \wedge P_2 \rightarrow \bot$.

The graph corresponding to $\mathcal{P}$ is given in Figure 6.2. It is also evident that, given that a pebbling of $\bot$ from $\{\top\}$ cannot be derived, the Horn formula $\mathcal{P}$ is satisfiable. The addition of a clause $P_3 \rightarrow P_2$ to $\mathcal{P}$ would, instead, cause its unsatisfiability. $\Diamond$

In order to test the satisfiability of a Horn formula, the D&G algorithm tests separately the satisfiability of each propositional letter P, by adding an arc from $\top$ to P, and testing if a pebbling exists of $\bot$ from $\{\top\}$. In such case, the current letter P leads to the unsatisfiability of the formula, and no other tests need to be runned. Each test is independent from the others, in the sense

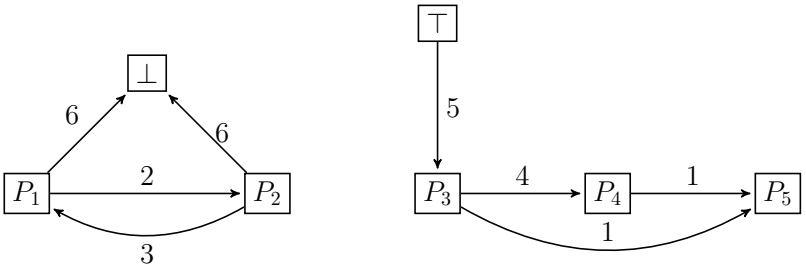


Figure 6.2: Example of graph associated to a Horn formula.

that they always start from a copy of the basic graph encoding (*i.e.*, the additional arc from $\top$ to the current letter is not permanently added).

In our approach, in order to reduce the conservativity principle problem to a consistency principle repair problem, we will introduce disjointess clauses in the Horn projection of the input ontologies. One requirement will be to avoid additional unsatisfiabilities unrelated to the conservativity violations (that is, unsatisfiabilities in the Horn projections of the input ontologies, which are assumed to be satisfiable). To this aim, we need to define safety conditions for disjointess clauses addition to a satisfiable Horn propositional formula (given in Definition 6.2).

Proposition 6.2. Given a satisfiable Horn propositional formula $\mathcal{P}$, the addition of a (*j-th*) disjointess clause $A \wedge B \rightarrow \bot$ will not cause any (potential) unsatisfiability of the letters of $\mathcal{P}$ iff:

(i) neither $\mathcal{P} \models A \rightarrow B$, nor $\mathcal{P} \models B \rightarrow A$ holds,

(ii) there exists no letter C such that $\mathcal{P} \models C \rightarrow A$ and $\mathcal{P} \models C \rightarrow B$.

Proof. First of all we need to prove that the addition of the *j-th* clause in the two aforementioned cases actually causes the unsatisfiability of $\mathcal{P}$. This is evident by observing Figures 6.3 and 6.4, respectively. In both cases we assume by hypothesis that a pebbling of A and B from $\{\top\}$ exists (in the second case through C). At this point the pebbling of $\bot$ from $\{\top\}$ immediately follows from the application of the (*j-th*) added disjointness clause. Note also that case 1 is a special case of 2, assuming $C = A$ (there is always a pebbling of a letter from its singleton, by Definition 6.3).

Assume that there exists another configuration, different from the previous ones, in which the disjointess clause addition causes an unsatisfiability for a letter D. To this aim, there must be a pebbling of both A and B, from a set X such that $D \in X$ (otherwise we are in the trivial case where $\mathcal{P} \models \top \rightarrow A$ and $\mathcal{P} \models \top \rightarrow B$, but this case is prevented by condition (i)).

Given that such pebbling of both letters from X exists, two subsets of X also exist, namely $X_C = \{C_i \mid i \in [1 \ldots n]\}$ and $X_E = \{E_i \mid i \in [1 \ldots m]\}$, with $n, m > 0$ and any letter distinct from D, such that a pebbling of A from $X_C \cup \{D\}$ exists, and a pebbling of B from $X_E \cup \{D\}$

exists. Given that, by hypothesis, the unsatisfiability is detected for letter D, through the inserted disjointness clause, it is evident that this letter must belong to both sets (the configuration is depicted in Figure 6.5, where the possible arcs between D and C_i/E_k, representing pebbling relation closure, are omitted for clarity).

Assume that this is not the case for X_C. Then, for each letter C_i of X_C, there must be a pebbling from $\{\top\}$ (that is, there must be a clause composed by the only positive literal C_i). Given that, by hypothesis, a pebbling of A from X_C exists, there is also a pebbling of A from $\{\top\}$. But again, in this case $\mathcal{P} \models \top \rightarrow A$ would hold, and condition (i) would have been violated, because $\mathcal{P} \models B \rightarrow A$ would trivially hold, independently from B.

Now, for each letter C_i and E_k, in order to be able to derive a pebbling for A and B from $\{\top\}$, either a pebbling from $\{D\}$ exists (and in this case these letters can be ignored because depending on the truth assignment of D), or a pebbling from $\{\top\}$ exists (and they again can be ignored, applying the semantic preserving *unit propagation* technique [DG84]). Since none of these two cases apply, the considered letter cannot belong to X, contradicting the hypothesis.

□

From the proof of Proposition 6.2, it is evident that this additional case reduces to that of condition (ii).

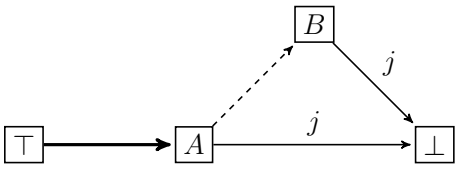


Figure 6.3: Horn unsatisfiability caused by disjointess clause addition between letters involved as head and body of a single clause. Bold arc is added for testing the satisfiability of the choosen letter. Dashed arcs represent pebbling relation closure.

Structural Indexing. Given that queries over the structural relationships of ontologies are heavily employed in our approach (both for violations detection, and for disjointess clauses safety check), we rely on the optimized structural index of LogMap [JRC11, JRMCH13], based on interval labelling schema techniques proposed by Agrawal *et al.* [ABJ89] for efficiently encoding the subsumption and disjointess relations of an OWL 2 ontology.

Specifically, the structural index exploits an optimized data structure for storing directed acyclic graphs (DAGs), and it allows us to answer many entailment queries over the concept hierarchy as an index lookup operation, and hence without the need of an OWL 2 reasoner (after the

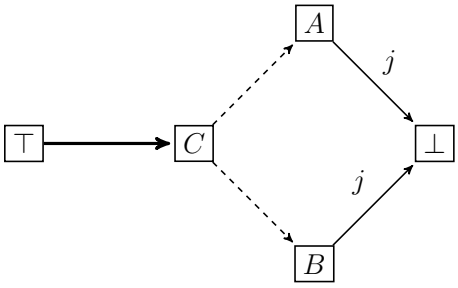


Figure 6.4: Horn unsatisfiability caused by disjointess clause addition between letters sharing a "descendant". Bold arc is added for testing the satisfiability of the choosen letter. Dashed arcs represent pebbling relation closure.

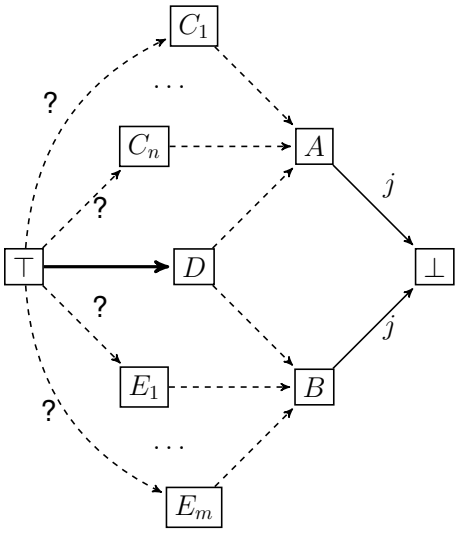


Figure 6.5: Generic Horn unsatisfiability scenario caused by disjointess clause addition. Bold arc is added for testing the satisfiability of the choosen letter. Dashed arcs represent pebbling relation closure.

initial classification of the ontology). This kind of index has demonstrated to significantly reduce the cost of answering taxonomic queries [CPST03, NB09] and disjointness relationships queries [JRC11, JRCZH12].

Definition 6.4, adapted from [ABJ89], formally defines the structural index.

Definition 6.4. Let $\mathcal{O}$ be an ontology classified using an OWL 2 reasoner, and let G be the DAG representing the subsumption hierarchy, between named classes, of $\mathcal{O}$ (the equivalent concepts are merged into a single node, called representative node).

1. Let G' be the graph obtained from G by reversing all its arcs.

2. Find a spanning tree T for G', called the tree-cover of G'.

3. Assign postorder numbers and indices to the nodes of T. Thus, at the end of this step, an interval $[i, j]$ would be associated with each node, such that j is the postorder number of the node, and i the lowest postorder number among its descendants.

4. Examine all the nodes of G in reverse topological order. At each node p, do the following processing:

   - For every arc (p, q), add all intervals associated with the node q to the intervals associated with node p.
   - At the time of adding an interval to the interval set associated with a node, if one interval is subsumed by another, discard the subsumed interval. That is, if the two intervals $[i_1, i_2]$ and $[j_1, j_2]$ are such that $i_1 \leq j_1$ and $i_2 \geq j_2$, then discard $[j_1, j_2]$.

$\triangle$

It can be proved [ABJ89] that a path from a node having descendant interval $[i, j]$ to a node having number k exists iff $i \leq k < j$. Therefore, by means of tree intervals, we have a compressed encoding of the reachability closure of a DAG, and reachability queries can be reduced to range comparisons. The information associated with a representative node is copied to all the concepts it includes.

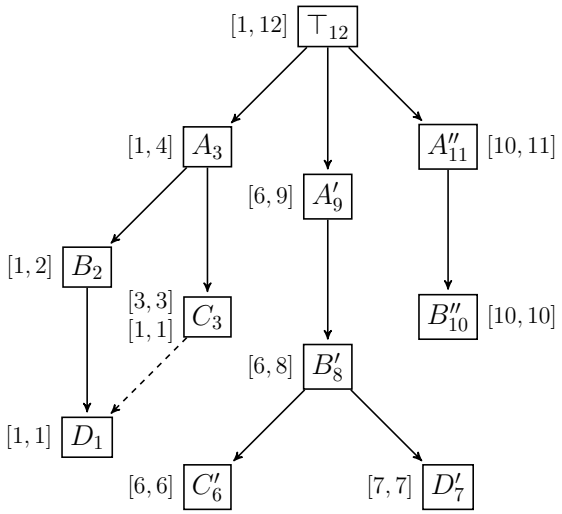


Figure 6.6: Classification DAG of the ontology (arcs are reversed). A possible tree-cover is represented by the removal of the dashed arc. Each node has its postorder number as subscript and associated tree interval(s).

We now describe, by means of Example 6.4, how the structural index is computed, and how taxonomical queries can be answered through it.

Operation	Description
getConcepts	get all the encoded concepts
shareDesc	tests if the concepts share at least a descendant concept
inSubRel	tests if the concepts are in subconcept relation
inSubSupRel	tests if the concepts are either in subconcept or superconcept relation
areDisj	tests if the concepts are disjoint
inEqRel	tests if the concepts are in equivalence relation
getEqConcepts	retrieves the concepts, if any, in the same representative node
getDirectSubConcepts	retrieves direct subconcepts w.r.t. the DAG
getOntoId	gets the id of the ontology the concept belongs to
sameOntology	compares the ontology identifier of the two concepts

Table 6.4: Operations supported by the structural index.

Example 6.4. Consider the classification DAG of a given ontology $\mathcal{O}$ such that its arc reversal leads to the DAG G' shown in Figure 6.6. Assume that the chosen tree-cover of G' coincides with G', exluding the dashed arc (the only alternative tree-cover would have been to include the dashed arc (C, D) and to ignore the arc (B, D)). The subscript of each node represents its postorder number. In addition, the tree intervals for each node are also represented. At this point, the only interval that the reverse topological visit of G' would add is $[1, 1]$ (to node C), because it cannot be merged, nor it is subsumed, by the existing interval $[3, 3]$ of C. Taxonomical queries can now be answered. For instance, in order to test if $\mathcal{O} \models A'' \sqsubseteq A'$, it suffices to check if the postorder number of A'' is included in any of the intervals of A'. Given that $11 \notin [6, 9]$, we know that $\mathcal{O} \not\models A'' \sqsubseteq A'$. Instead, from the fact that $7 \in [6, 9]$, we know that $\mathcal{O} \models D' \sqsubseteq A'$. $\Diamond$

Starting from the available disjointness axioms, an analogous labelling schema is proposed for disjointess between (named) concepts, exploiting the characteristic that if $\mathcal{O} \models A \sqsubseteq \neg B$, then also $\mathcal{O} \models A' \sqsubseteq \neg B$ and $\mathcal{O} \models A \sqsubseteq \neg B$ hold, with A' (resp. B') any subconcept of A (resp. B).

Additionally, the structural index implemented in LogMap is enriched with a labelling schema encoding the set of ancestors/descendants, and support the whole interval's algebra proposed by Berlanga *et al.* [NB09] (in turns based on the approach of Agrawal *et al.*).

The structural index also keeps track, for each concept, of any explictly asserted disjointness axiom, and the direct equivalence/subclass/superclass axioms (w.r.t. the DAG, not w.r.t. the asserted axioms). The operations supported by the index that are relevant for our approach are reported in Table 6.4.

As shown in the example, the $\bot$ concept is excluded, and it is therefore not considered by the implemented operations (otherwise the result of shareDesc function would always be trivially true).

Mapping Repair. As already discussed, the addition of enough disjointness clauses to the Horn projections of the input ontologies allows us to reduce the conservativity principle violations repair problem to a consistency repair problem. To this aim, our algorithm concretely employs the mapping (incoherence) repair algorithm of LogMap.

The mapping repair process exploits the D&G algorithm for propositional Horn satisfiability and checks, for every propositional letter A of the formula $\mathcal{P}$, the satisfiability of the propositional formula $\mathcal{P}_A = \mathcal{P} \cup \{\top \rightarrow A\}$. Satisfiability of $\mathcal{P}_A$ is checked in worst-case linear time in the size of $\mathcal{P}_A$, and the number of D&G calls is also linear in the number of propositional letters in $\mathcal{P}$. In case of unsatisfiability, the variant of the algorithm implemented in LogMap also allows us to record *conflicting* mappings involved in the unsatisfiability, which will be considered for the subsequent repair process.

The unsatisfiability will be fixed by removing some of the identified mappings. To this aim, the algorithm aims at selecting an approximation of the minimal hitting set (w.r.t. total mapping confidence) between the set of conflicting mappings, for the different unsatisfiabilities that are detected. In case of multiple options, the mapping confidence will be used as a differentiating factor.[3]

A full example of mapping repair will be given in Example 6.8, Section 6.4, after discussing the reduction of the conservativity problem to a consistency repair problem.

The proposed technique allows to solve unsatisfiabilities in the Horn propositional projection of the aligned ontology. This feature on one hand favors scalability of the technique, but on the other cannot guarantee completeness. In Section 6.5, we experimentally verify the efficiency of the approach, and that the number of unsolved violations, due to the incompletness of the repair technique, is extremely limited.

DL Modules. The notion of *module* has been introduced in DLs as a general technique for ensuring scalability of ontology-based algorithms. The main intuition behind this notion is that a module for a given ontology $\mathcal{O}$ w.r.t. a seed signature Σ, is simply a subset of $\mathcal{O}$ that preserves the entailment relation over the axioms expressed using Σ. In Definition 6.5, modules are formally introduced.

Definition 6.5. A subset $\mathcal{O}'$ of $\mathcal{O}$ is a *module* of $\mathcal{O}$ w.r.t. a *seed* signature Σ (denoted also as $module(\mathcal{O}, \Sigma)$) iff for any axiom α such that $Sig(\alpha) \subseteq \Sigma$, $\mathcal{O} \models \alpha \iff \mathcal{O}' \models \alpha$. $\triangle$

Due to the high cost of computing strictly minimal modules and undecidability of module computation for simple sublanguages (see Section 4.6.2), different approximations (both semantical and syntactical) have been proposed in literature, such as *locality-based modules* [CHKS07,

[3] In scenarios where the confidence of the mapping is missing (*e.g.*, in reference or manually created mapping sets) or unreliable, our mapping repair technique computes fresh confidence values based on the locality principle [JRCHB11a].

CHKS08, GHWKS10], covering up to $\mathcal{SROIQ}$ ontologies [GHWKS10]. These approximations do not compute minimal modules, but they guarantee the *coverage* property for the entailment relation over the seed signature (notion at the basis of Definition 6.5).

Different techniques have been proposed for computing locality-based modules such as $\bot$-locality and $\top$-locality modules. Other variants have been conceived [SSZ09], but given that in this work we will only employ $\bot$ locality-based modules (simply modules, when not specified), they will not be covered.

We report below two propositions for $\bot$-locality modules from [GHWKS10] and [Tsa14], respectively, that will be used in Section 6.4 for proving the correctness of one of the proposed optimizations. These propositions have been already used for implementing incremental reasoning in [GHWKS10] and in the OWL 2 reasoner *FaCT++* [Tsa14].

More specifically, Proposition 6.3 (Proposition 3 in [GHWKS10]) tells us that, given a subsumption between named classes, the module computed using as seed the singleton of the subsumee, is a module for such axiom. Proposition 6.3 has an interesting corollary stating that, for each subsumption axiom between named classes, the subsumer always belongs to the signature of the module of the subsumee. This corollary, in conjunction with Proposition 6.5 (Proposition 3 in [Tsa14]), proves that the module of the subsumer can be computed starting from the module of its subsumee.

In place of $module(\mathcal{O}, \Sigma)$, we sometimes employ the compact notation $\mathcal{O}_\Sigma$.

Proposition 6.3. Let $\mathcal{O}$ be an ontology, α an axiom in a form $A \sqsubseteq B$, where $A, B \in Sig(\mathcal{O})$ are concept names. Then, $\mathcal{O}^\alpha$ can be computed as $\mathcal{O}^A$, which is a $\bot$-locality-based module for signature $\{A\}$. □

Corollary 6.4. Let $\mathcal{O}$ be an ontology, and let A, B be concept names. If $\mathcal{O}_A \models A \sqsubseteq B$, then $B \in Sig(\mathcal{O}_A)$. □

Proposition 6.5. Let A and B be two concept names such that $B \in Sig(\mathcal{O}_A)$. Then, $\mathcal{O}_B \subseteq \mathcal{O}_A$. In other words, $\mathcal{O}_B = module(\mathcal{O}_A, \{B\})$. □

At this point we are ready to state Proposition 6.6 and Proposition 6.7, that will be key for the completeness of our optimization.

Proposition 6.6. Given an ontology $\mathcal{O}$, and a set of subsumption axioms S of $\mathcal{O}$, let $\Sigma = \{A \mid A \sqsubseteq B \in S\}$. The module $\mathcal{O}'$ extracted using as seed the signature Σ guarantees that: for each $C \in N_C(\mathcal{O})$ and $A \in \Sigma$ such that $\mathcal{O} \models A \sqsubseteq C$, $\mathcal{O}' \models A \sqsubseteq C$ holds.

Proof. Assume, by contradiction, that $\mathcal{O}' \not\models A \sqsubseteq C$. We know, from Proposition 6.3, that the module $\mathcal{O}_A$ computed using as seed the signature $\{A\}$ is a module for any subsumption axiom having A as subsumee. Given that $\Sigma \subseteq Sig(\mathcal{O}')$ holds, from Proposition 6.5, for each concept

$A \in \Sigma$, $\mathcal{O}'_A \subseteq \mathcal{O}'$. Therefore, from monotonicity of standard DLs, we have a contradiction, because $\mathcal{O}'_A \models A \sqsubseteq C \implies \mathcal{O}' \models A \sqsubseteq C$.

□

Proposition 6.7. Given an ontology $\mathcal{O}$, and a set of subsumption axioms S of $\mathcal{O}$, let $\Sigma = \{A \mid A \sqsubseteq B \in S\}$, and let $\Sigma' = \{B \mid A \sqsubseteq B \in S\}$. The module $\mathcal{O}'$ extracted using as seed the signature Σ guarantees that: for each $C \in N_C(\mathcal{O})$ such that $B \sqsubseteq C$, for some $B \in \Sigma'$, $\mathcal{O}' \models B \sqsubseteq C$ holds.

Proof. The proof follows from Corollary 6.4, because given such concept $B \in \Sigma'$, $B \in Sig(\mathcal{O}')$, due to the subsumption in S and that we used Σ as seed signature. Proposition 6.5 ensures that, whenever $B \in Sig(\mathcal{O}')$ holds, $module(\mathcal{O}', \{B\}) \subseteq \mathcal{O}'$. By definition, module $\mathcal{O}'_B$ must preserve any subsumption having B as subsumee (by Proposition 6.3), and therefore also $B \sqsubseteq C$, as desired.

□

From Proposition 6.7, it is evident that the module computed in this way allows us to preserve and reason over chains of subsumptions starting from elements in the seed signature.

Example 6.5 provides an instance of $\bot$-locality module extraction.

Example 6.5. Consider the aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$ of Section 6.3.1. The module $module(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}, \{Company_1\})$ is equivalent to the following set of axioms:

- $Company_1 \equiv Company_2$,
- $Company_2 \sqsubseteq Field\_operator_2$,
- $Field\_operator_2 \sqsubseteq Operator_1$,
- $Field\_operator_2 \sqcap Owner_2 \sqsubseteq Field\_owner_2$,
- $Owner_1 \equiv Owner_2$,
- $Operator_1 \sqsubseteq Owner_1$,
- $Operator_1 \sqsubseteq Company_1$,
- $Field\_owner_2 \sqsubseteq Owner_2$.

By analyzing the extracted module and the classification DAG of the aligned ontology shown in Figure 6.7, it is evident that the desired property of preserving the "subsumption chains" starting at the seed(s) (for ontologies closed by classification by means of an OWL 2 resoner) is guaranteed by $\bot$-locality modules. ◊

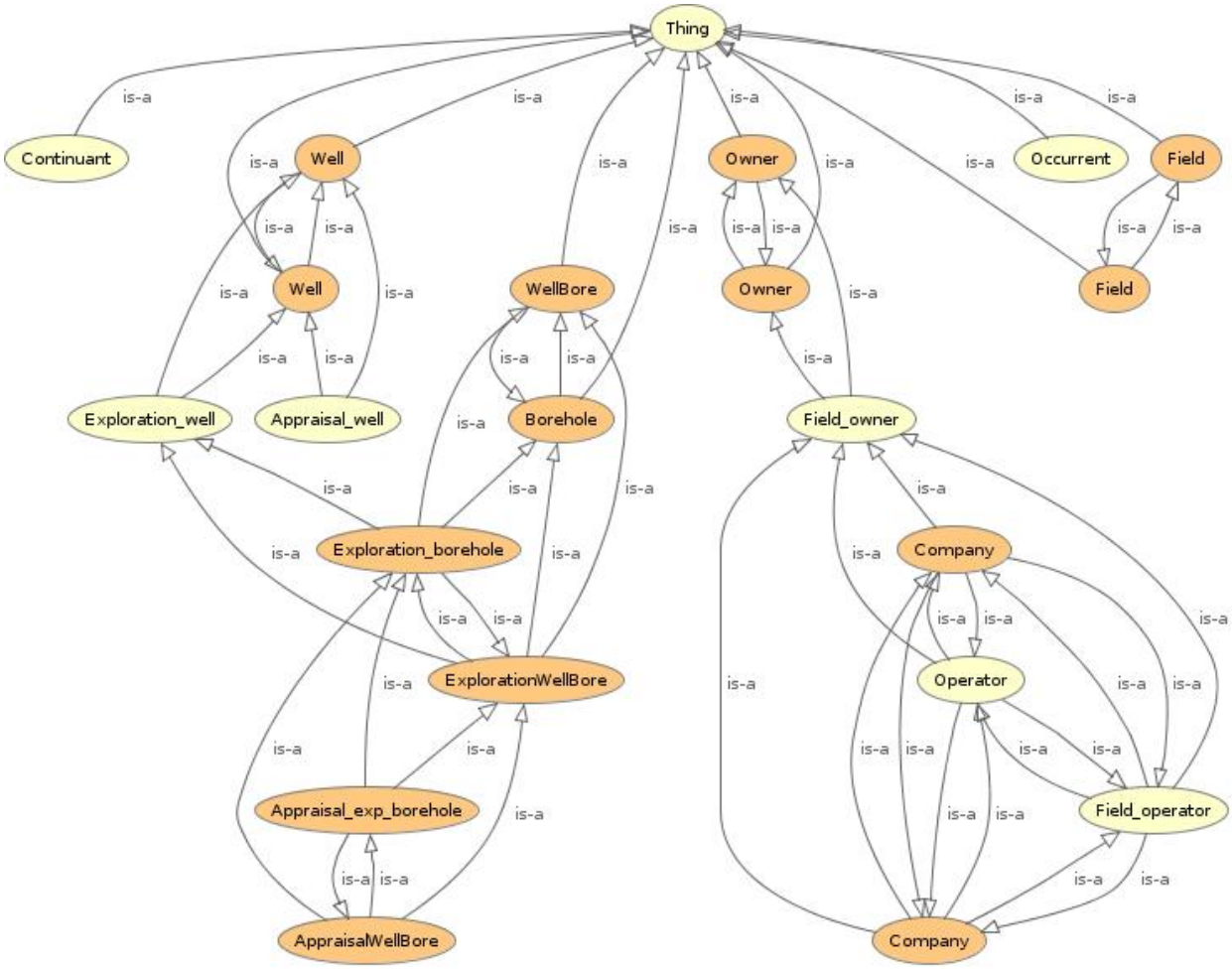


Figure 6.7: Classification DAG of the aligned ontology of Section 6.3.1.

Direct Conservativity Subsumption Violations. As it will be empirically observed in Section 6.5, the number of (basic) subsumption violations can be overwhelming for the user. Not all these violations are, however, independent from the others. For this reason, in order to reduce the number of violations presented to the user, the notion of *direct violations* (opposed to *derived violations*) is formally introduced in Definition 6.6.

Definition 6.6. Let $\alpha = A \sqsubseteq B$ be a subsumption conservativity principle violation affecting an aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$, where $A, B \in N_C(\mathcal{O}_i)$, and $i \in \{1, 2\}$. α is said to be a *direct violation* iff at least one of the following conditions holds:

(i) no another violation $A' \sqsubseteq B'$ exists such that $\mathcal{O}_i \models A \sqsubseteq A'$ and $\mathcal{O}_i \models B' \sqsubseteq B$,

(ii) two mappings $\langle A, C_1, \sqsubseteq, \_\rangle$, $\langle C_n, B, \sqsubseteq, \_\rangle$ exist in $\mathcal{M}$, and $\mathcal{O}_j \models C_1 \sqsubseteq C_n$, with $j \in \{1, 2\}$ and $i \neq j$.

A subsumption violation that is not direct is called *derived*. △

Example 6.6 provides an example of direct and derived violation.

Example 6.6. Consider the input ontologies and mappings of Tables 6.1 and 6.2, respectively. Figures 6.8 and 6.9 show the classification DAG associated with ontology $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively. Among the identified violations (listed in Table 6.3), σ_3 is an example of direct one (no named superconcepts can be derived for $Field\_operator$ in $\mathcal{O}_2$, and therefore the first condition is trivially satisfied). σ_7, instead, is a derived violation, due to σ_3. The first condition is satisfied, because $\mathcal{O}_2 \models Company \sqsubseteq Field\_operator$ and $\mathcal{O}_2 \models Field\_owner \sqsubseteq Field\_owner$ hold. Note that, despite the two violations need to be distinct, they can share an element, as shown above. Concerning the second condition, it cannot be satisfied because despite the existence of mapping $m_8 = \langle Company_1, Company_2, \equiv, 0.9\rangle$, $Company$ has no named superconcepts in (the classification of) ontology $\mathcal{O}_1$, and no other mappings of the form $\langle Company_1, \_, \sqsubseteq, \_\rangle$ exist in $\mathcal{M}$. ◊

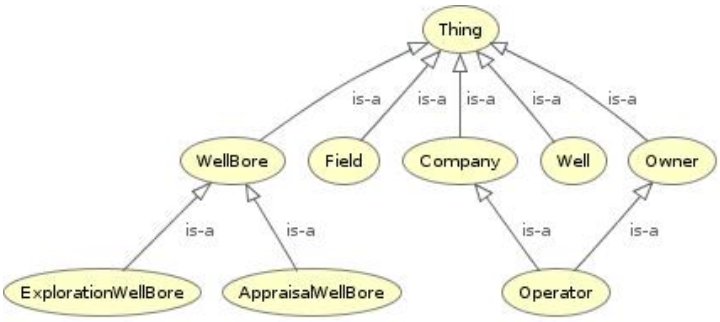


Figure 6.8: Classification DAG of the input ontology $\mathcal{O}_1$ of Section 6.3.1.

6.4 Algorithms

The main ingredient of our proposal for solving basic conservativity principle violations is the (partial) reduction of the conservativity principle repair problem to a mapping (incoherence) repair problem, through the assumption of disjointness. Currently, our method reuses and adapts the structural indexing and reasoning techniques implemented in LogMap [JRC11]. However, alternative mapping repair systems could be used in our solution, such as ALCOMO [Mei11] or AML [FPS+13]. Note that, to the best of our knowledge, these mapping repair systems have only focused on solving violations of the consistency principle.

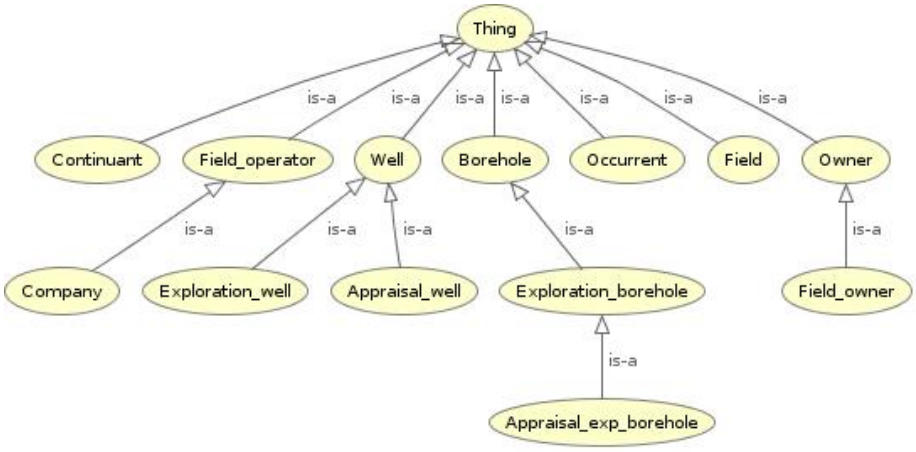


Figure 6.9: Classification DAG of the input ontology $\mathcal{O}_2$ of Section 6.3.1.

Algorithm 16 shows the pseudocode of the proposed method for detecting and correcting subsumption conservativity violations, named *SubRepair* algorithm. Before describing the algorithm we detail two preprocessing steps, module extraction and consistency repair, performed before invoking it.

Module Extraction. In order to reduce the size of the problem, we extract two $\bot$-locality-based modules (introduced in Section 6.3.2), one for each input ontology, using the entities involved in the mappings $\mathcal{M}$ as seed signatures for the module extractor. These modules preserve the semantics for the given entities, can be efficiently computed, and are typically much smaller than the original ontologies.

Full Consistency Repair. As discussed in Section 4.3.6, our approach expects to work on coherent alignments. In order to meet this requirement we perform a preliminary consistency repair, as a two-step process. Firstly, the consistency repair facility of LogMap is used to repair the aligned ontology. Then, any unsolved incoherence is detected through an OWL 2 reasoner, and solved by computing a single justification [Hor11], and removing the mapping with least confidence appearing in it. This last step is iteratively applied until no more incoherencies are present in the aligned ontology. This consistency repair process is sound and complete, but it does not guarantee the optimial minimization of the total confidence of the removed mappings. However, for the purpose of our approach we only need to fully repair the input alignment, and the possible bias introduced by previous repair process is taken into account in the experimental evaluation of Section 6.5.

In the remaineder, if not explicitly stated, we assume that our methods receive as input modules (computed starting from the input ontologies and the given alignment), and coherent alignments

Algorithm 16 SubRepair algorithm to detect and solve conservativity principle violations w.r.t. subsumption

Input: $\mathcal{O}_1, \mathcal{O}_2$: input ontologies; $\mathcal{M}$: input mappings; $optimization$,$direct$,$useDFS$: Boolean flags
Output: $\mathcal{M}'$: output mappings; $\mathcal{R}^{\approx}$: approximate repair; $disj$: number of disjointness rules

1: $\langle \mathcal{P}_1, \mathcal{P}_2 \rangle \leftarrow$ propositionalEncoding$(\mathcal{O}_1, \mathcal{O}_2)$
2: $SI_1 \leftarrow$ structuralIndex$(\mathcal{O}_1)$
3: $SI_2 \leftarrow$ structuralIndex$(\mathcal{O}_2)$
4: **if** $(optimization =$ true**) then**
5: $\quad \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \leftarrow \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$ ▷ The aligned ontology is computed
6: $\quad SI_{\mathcal{U}} \leftarrow$ structuralIndex$(\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M})$
7: $\quad \langle {\mathcal{P}_1}^d, disj_1 \rangle \leftarrow$ disjointAxiomsExtensionOptimized$(\mathcal{P}_1, SI_1, SI_{\mathcal{U}}, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}, direct, useDFS)$ ▷ See Algorithm 18
8: $\quad \langle {\mathcal{P}_2}^d, disj_2 \rangle \leftarrow$ disjointAxiomsExtensionOptimized$(\mathcal{P}_2, SI_2, SI_{\mathcal{U}}, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}, direct, useDFS)$
9: **else**
10: $\quad \langle {\mathcal{P}_1}^d, disj_1 \rangle \leftarrow$ disjointAxiomsExtensionBasic$(\mathcal{P}_1, SI_1)$ ▷ See Algorithm 17
11: $\quad \langle {\mathcal{P}_2}^d, disj_2 \rangle \leftarrow$ disjointAxiomsExtensionBasic$(\mathcal{P}_2, SI_2)$
12: **end if**
13: $\langle \mathcal{M}', \mathcal{R}^{\approx} \rangle \leftarrow$ mappingRepair$({\mathcal{P}_1}^d, {\mathcal{P}_2}^d, \mathcal{M})$ ▷ See Algorithm 2 in [JRMCH13]
14: $disj \leftarrow disj_1 + disj_2$
15: **return** $\langle \mathcal{M}', \mathcal{R}^{\approx}, disj \rangle$

(obtained using the full repair detailed above). When no confusion arises, we directly refer to these modules as the input ontologies.

Specifically, the SubRepair algorithm accepts as input two (modules computed from the input) OWL 2 ontologies, $\mathcal{O}_1$ and $\mathcal{O}_2$, and a set of (coherent) mappings $\mathcal{M}$. Additionally, an optimized variant to add disjointness axioms can be selected. $direct$ Boolean flag controls the use of all the subsumption violations or direct violations only. The explanation of $useDFS$ flag is postponed to the description of Algorithm 21, which is the only algorithm influenced by this flag.

The SubRepair algorithm outputs the number of disjointness clauses added during the process $disj$, a set of mappings $\mathcal{M}'$, and an (approximate) repair $\mathcal{R}^{\approx}$ such that $\mathcal{M}' = \mathcal{M} \setminus \mathcal{R}^{\approx}$. The (approximate) repair $\mathcal{R}^{\approx}$ aims at solving most of the conservativity principle violations of $\mathcal{M}$ with respect to $\mathcal{O}_1$ and $\mathcal{O}_2$. In the following paragraphs we describe the techniques used at each step.

Propositional Horn Encoding. The ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ are encoded as the Horn propositional formulas, $\mathcal{P}_1$ and $\mathcal{P}_2$ (line 1 in Algorithm 16), as described in Definition 6.1. For example, the concept hierarchy provided by an OWL 2 reasoner (*e.g.*, [MSH09, KKS11]) will be encoded as $A \rightarrow B$ rules, while the explicit disjointness relationships between concepts will be represented as $A_i \wedge A_j \rightarrow \bot$. Note that the input mappings $\mathcal{M}$ can already be seen as propositional implications. As already discussed, this encoding is key to the mapping repair process.

Structural Indexing. The concept hierarchies provided by an OWL 2 reasoner and the explicit disjointness axioms of $\mathcal{O}_1$ and $\mathcal{O}_2$ are efficiently indexed into the structural index introduced in Definition 6.4 (lines 2 and 3 in Algorithm 16).

Algorithm 17 disjointAxiomsExtensionBasic function for basic disjointness axioms extension

Input: $\mathcal{P}$: propositional theory; SI: structural index
Output: $\mathcal{P}^d$: extended propositional theory; $disj$: number of disjointness rules

```
1: disj ← 0
2: P^d ← P
3: for each pair ⟨A, B⟩ in orderedVariablePairs(P) do
4:     if not (SI.areDisj(A, B) or SI.inSubSupRel(A, B) or SI.shareDesc(A, B)) then
5:         P^d ← P^d ∪ {A ∧ B → false}
6:         SI ← SI.updateIndex(A ⊓ B → ⊥)
7:         disj ← disj + 1
8:     end if
9: end for
10: return ⟨P^d, disj⟩
```

Disjointness Addition. In order to reduce the conservativity problem to a mapping incoherence repair problem, following the *assumption of disjointness* notion, we need to automatically add sufficient disjointness axioms into each ontology $\mathcal{O}_i$.

However, the insertion of additional disjointness axioms δ may lead to unsatisfiable classes in $\mathcal{O}_i \cup \{\delta\}$, as shown in Example 6.7.

Example 6.7. Consider the axiom β_9 from Table 6.1. Following the *assumption of disjointness* a very naïve algorithm would add disjointness axioms between $Borehole$, $Continuant$ and $Occurrent$, which would make $Borehole$ unsatisfiable. ◊

In order to detect if each candidate disjointness axiom leads to an unsatisfiability, a non naïve algorithm requires to make an extensive use of an OWL 2 reasoner to check if there are new unsatisfiable classes in $\mathcal{O}_i \cup \{\delta\}$. In large ontologies, however, such extensive use of the reasoner may be prohibitive.

Our method, in order to address this issue, exploits the propositional encoding and structural indexing of the input ontologies. Thus, checking if $\mathcal{O}_i \cup \{\delta\}$ contains unsatisfiable classes is restricted to the Horn propositional case. Specifically, the graph encoding of the D&G algorithm, (Definition 6.2), and the pebbling test (Definition 6.3) are employed.

We have implemented two algorithms to extend the propositional formulas $\mathcal{P}_1$ and $\mathcal{P}_2$ with disjointness rules of the form $A \wedge B \rightarrow \bot$ (see lines 4-12 in Algorithm 16). These algorithms guarantee that, for every propositional letter A in the extended propositional formula $\mathcal{P}_i^d$ (with $i \in \{1, 2\}$), the formula $\mathcal{P}_i^d \cup \{\top \rightarrow A\}$ is satisfiable. Note that this does not necessarily hold if the disjointness axioms are added to the OWL 2 ontologies, $\mathcal{O}_1$ and $\mathcal{O}_2$, as discussed above.

Algorithm 17 presents a (basic) algorithm to add as many disjointness rules as possible, for every pair of propositional letters A, B in the propositional theory $\mathcal{P}$ given as input. In order to minimize the number of necessary disjointness rules, the letters in $\mathcal{P}$ are ordered in pairs following a top-down approach. The algorithm exploits the structural index SI to check if two propositional letters (*i.e.*, concepts in the input ontologies) are disjoint (SI.areDisj(A, B)), they keep a sub/super-class relationship (SI.inSubSupRel(A, B)), or they share a common descendant

Algorithm 18 disjointAxiomsExtensionOptimized function for optimized disjointness axioms extension

Input: $\mathcal{P}$: propositional theory; SI: structural index; $SI_{\mathcal{U}}$: structural index of the aligned ontology; $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$: aligned ontology; $direct$,$useDFS$: Boolean flags
Output: $\mathcal{P}^d$: extended propositional theory; $disj$: number of disjointness rules

1: $disj \leftarrow 0$
2: $\mathcal{P}^d \leftarrow \mathcal{P}$
3: **for each** $A \rightarrow B$ **in** conservativityViolations$(SI, SI_{\mathcal{U}}, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}, direct, useDFS)$ **do** ▷ See Algorithm 19
4: **if not** $(SI.\text{areDisj}(A, B))$ **then**
5: $\mathcal{P}^d \leftarrow \mathcal{P}^d \cup \{A \wedge B \rightarrow \text{false}\}$
6: $SI \leftarrow SI.\text{updateIndex}(A \sqcap B \rightarrow \bot)$
7: $disj \leftarrow disj + 1$
8: **end if**
9: **end for**
10: **return** $\langle \mathcal{P}^d, disj \rangle$

$(SI.\text{shareDesc}(A, B))$ (line 4 in Algorithm 17). The above tests, performed using the structural index, enforce the safety conditions required by Definition 6.2, for avoiding undesired unsatisfiabilities in the projections of the input ontologies. The structural index is also updated to take into account the new disjointness rules (line 6 in Algorithm 17).

The addition of disjointness rules in Algorithm 17, however, may be prohibitive for large ontologies (see Section 6.5). Intuitively, in order to reduce the number of disjointness axioms, one should only focus on the cases where a conservativity principle violation occurs in the aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$, with respect to one of the input ontologies $\mathcal{O}_i$ (with $i \in \{1, 2\}$); *i.e.*, adding a disjointness axiom between each pair of classes $A, B \in \mathcal{O}_i$ such that $A \sqsubseteq B \in \text{basicViol}(\mathcal{O}_i, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$, as in Definition 4.26. Algorithm 18 implements this idea for the Horn propositional case and extensively exploits the structural indexing to identify the conservativity principle violations (called at line 3 of Algorithm 18, and detailed in Algorithm 19).

This algorithm also requires to compute the structural index of the integrated ontology, and thus its classification with an OWL 2 reasoner (line 6 in Algorithm 16). The classification time of the integrated ontology is known to be typically much higher than that of the input ontologies individually [JRCH12]. However, this was not a bottleneck in our experiments, as shown in Section 6.5.

Conservativity Principle Violations Detection. In Algorithm 19 the conservativity principle violations detection method is presented. The algorithm is meant to compute the conservativity violations for a single input ontology at a time, for this reason it takes as input the structural index of the considered input ontology $\mathcal{O}_i$ (namely SI), that of the aligned ontology (namely $SI_{\mathcal{U}}$), and a Boolean flag $direct$ controlling the kind of violations to detect, between all subsumption violations, or direct violations only (Definition 6.6). The need for the aligned ontology and the Boolean flag $useDFS$ will be motivated later.

At this point, for each concept C of $\mathcal{O}_i$, the getSubEqConcepts function is called at line 3. The

Algorithm 19 conservativityViolations function for conservativity principle violations detection

Input: SI: structural index; $SI_{\mathcal{U}}$: structural index of aligned ontology; $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$: aligned ontology; $direct$,$useDFS$: Boolean flags
Output: $viols$: set of detected violations

1: $viols \leftarrow \emptyset$
2: **for each** C **in** SI.getConcepts() **do**
3: **for each** C' **in** getSubEqConcepts($SI_{\mathcal{U}}, C, direct$) **do** ▷ Get all the candidates for violations, see Algorithm 20
4: **if not** SI.inSubRel(C', C)) **then**
5: $viols$.add($C' \sqsubseteq C$)
6: **end if**
7: **end for**
8: **end for**
9: **if** $direct$ **then**
10: $allSubViols \leftarrow$ conservativityViolations($SI, SI_{\mathcal{U}}, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$, false, $useDFS$) ▷ Detect all the subsumption violations
11: $violsChk \leftarrow allSubViols \setminus viols$ ▷ Candidates are subsumption violations not already known to be direct
12: $dirViols \leftarrow$ graphDirViolCheck($\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}, SI_{\mathcal{U}}, violsChk, useDFS$) ▷ See Algorithm 21
13: $viols$.addAll($dirViols$)
14: **end if**
15: **return** $viols$

function provides the subconcepts of C in the aligned ontology, which originally belong to the signature of $\mathcal{O}_i$. The different behaviour enforced by the flag $direct$ is practically implemented by the getSubEqConcepts function, as discussed next.

Therefore, the concepts returned by this function are the only candidates for being involved, as subsumees, in a violation with concept C. A candidate is actually involved in a violation if the subsumption relation was not holding in the input ontology, and this is tested for each subconcept C' at line 4 by means of the structural index SI. If the test is successful, the violation $C' \sqsubseteq C$ is added to the set of detected violations, $viols$. When all the concepts of $\mathcal{O}_i$ have been processed, the set of violations is returned (line 15).

The additional computation needed for ensuring a *complete* detection algorithm for direct violations (lines 9–14), is discussed after the description of the getSubEqConcepts function, shown in Algorithm 20, as its behaviour is propedeutic to the understanding of the incompleteness source.

Conservativity Violations Candidate Selection. The getSubEqConcepts function, takes as input the structural index for the aligned ontology, and the concept C for which we aim at computing the sub/equivalent concept(s), and a Boolean flag stating whether the result is to be used for computing direct violations or not.

In order to be able to cover both cases (*i.e.*, all subsumption violations or direct ones only) within a single algorithm, the visit is always performed in a top-down fashion, starting from the $\top$ concept, and all its equivalent classes, if any. For this reason, the queue of concepts to be analyzed is initialized with the concepts equivalent to $\top$ (line 3), and any further addition to the queue is assumed to be made at its tail, while the poll function retrieves (and removes) from the head of the queue.

In the case in which we do not seek for direct violations, we avoid visiting multiple times a single

Algorithm 20 getSubEqConcepts function for detecting violation's candidates

Input: $SI_{\mathcal{U}}$: structural index of the aligned ontology; C: named concept; $direct$: Boolean flag
Output: $subEq$: set of sub/equivalent concept(s)

```
1: subEq ← ∅
2: visited ← ∅
3: Q ← {T' | T' = ⊤ ∨ SI_U.inEqRel(⊤, T')}
4: while not Q.isEmpty() do
5:     C' ← Q.poll()
6:     if not direct then
7:         if visited.contains(C') then
8:             continue
9:         end if
10:        visited.add(C')
11:    end if
12:    isEquivTop ← SI_U.inEqRel(⊤, C')
13:    sameOnto ← SI_U.sameOntology(C, C')
14:    if sameOnto then
15:        subEq.add(C')
16:    end if
17:    eqNotSame ← false
18:    if not isEquivTop then
19:        for each C'' in SI_U.getEqConcepts(C') do
20:            if SI_U.sameOntology(C, C'') then
21:                subEq.add(C'')
22:            else
23:                eqNotSame ← true
24:            end if
25:        end for
26:    end if
27:    if (direct and eqNotSame) or (not direct and sameOnto) or isEquivTop then
28:        Q.addAll(SI_U.getDirectSubConcepts(C'))
29:    end if
30: end while
31: return subEq
```

concept (line 7), and we keep track of the visited ones using the $visited$ set, updated at line 10. The top-down visit allows us to apply a "blocking" technique, avoiding to add the subconcepts of any concept belonging to the same input ontology of C, because they cannot be, by definition, involved in a direct violation (they always violate condition 1 of Definition 6.6). When we seek for all the subsumption violations, adding these classes to Q is not necessary, and it is avoided by testing their membership to the $visited$ set.

The visit is concretely performed by means of the *while* loop at lines 4–30, that is composed by the following steps.

At line 12 we keep track with the Boolean flag $isEquivTop$ if the concept C' is equivalent $\top$ or not, in order to avoid a loop over the named concepts equivalent $\top$, if they are more than one (test at line 4).

We remind that the employed structural index for the aligned ontology encodes its structural relationships, but nonetheless we aim at collecting the subconcepts and equivalent ones w.r.t. the input ontology the concept C belongs to, because only these concepts can appear in a violation. We use the Boolean flag $sameOnto$ (initialized at line 13) to keep track if C and the currently

considered concept C' belong to the same input ontology. In such case, given that we only add to queue Q subconcepts of C, we add C' to the set $subEq$ (line 14).

The *for* loop of lines 18–26 processes each equivalent concept C'' of C', and if it belongs to the same input ontology of C, it is added to the set $subEq$ (line 21), otherwise we keep track of the existence of an equivalent concept C'' that is not from the same input ontology of C, by setting the flag $eqNotSame$ to true (line 23).

Finally the last step of the *while* loop is at line 27, where if we are interested in direct violations (*i.e.*, $direct$ flag is true) and at least one among the equivalent classes of C', or C' itself, is from the other input ontology w.r.t. that of C, we add all their subconcepts to queue Q (that is, they become concepts to be analyzed). The reason is that any of these concepts which belongs to the same input ontology of C is a direct violation, given that we "reached" it through a path traversing only concepts belonging to the other input ontology. Therefore, condition 2 of Definition 6.6 is satisfied, even if condition 1 is not. Figure 6.10 provides an example of the aforementioned situation. In the example, a path from concept B to concept C exists due to the equivalence of classes A', A'' and A''' (note that we assume the existence of at least one path between C and these concepts, composed by elements of the first input ontology).

In addition, again at line 27, we add the subconcepts of C to queue Q even if C' is equivalent to $\top$ (*i.e.*, $isEquivTop$ flag is set to true), or if the considered concept C' belongs to the same ontology of C (*i.e.*, $sameOnto$ flag is true), and we are not seeking for direct violations only (*i.e.*, $direct$ flag is false).

When the queue Q is empty, we have processed all the necessary concepts, and the set $subEq$ is then returned as output (line 31).

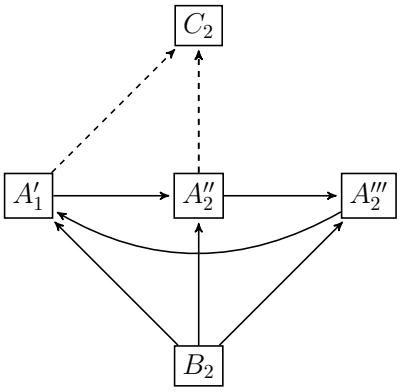


Figure 6.10: Example of direct violations involving equivalences. Concept's subscript denotes the input ontology it belongs to.

Direct Violations Detection in Practice. In order to be able to efficiently implement a detection method for detecting direct violations, we rely on an alternative definition w.r.t. to the one given in Definition 6.6, that differs in the formulation of the second condition. This definition makes use of the graph representation of Definition 5.2.

Definition 6.7. Let $\alpha = A \sqsubseteq B$ be a subsumption conservativity principle violation affecting an aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$, where $A, B \in N_C(\mathcal{O}_i)$, and $i \in \{1, 2\}$. α is said to be a *direct violation* iff at least one of the following conditions holds:

(i) no another violation $A' \sqsubseteq B'$ such that $\mathcal{O}_i \models A \sqsubseteq A'$ and $\mathcal{O}_i \models B' \sqsubseteq B$ exists,

(ii) a path $\pi = [A, C_1, \ldots, C_n, B]$ exists in $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$, with $n \geq 1$, $j \in \{1, 2\}$ and $i \neq j$, such that $C_k \in N_C(\mathcal{O}_j)$ for each $k \in [1 \ldots n]$. △

Proposition 6.8 proves the equivalence between the two definitions.

Proposition 6.8. The definitions for direct violations given in Definition 6.6 and Definition 6.7 are equivalent.

Proof. In order to prove the hypothesis, we only need to prove that the two formulations of the second condition are equivalent.
⇒: from Definition 5.2 we know that if a mapping $\langle E, F, \sqsubseteq, \_\rangle$ (or $\langle E, F, \equiv, \_\rangle$) exists in $\mathcal{M}$, then a corresponding arc $(E, F, \_)$ is created. For this reason arcs $(A, C_1, \_)$ and $(C_n, B, \_)$ belong to the set of arcs of $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$. From Definition 5.2, if $\mathcal{O}_j \models C_1 \sqsubseteq C_n$ holds, a path $[C_1, \ldots, C_n]$ in $G(\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})$ must exist.
⇐: from Definition 5.2 we know that each arc $(E, F, \_)$ involving named concepts from different input ontologies is created iff a mapping $\langle E, F, \sqsubseteq, \_\rangle$ (or $\langle E, F, \equiv, \_\rangle$) exists in $\mathcal{M}$. Definition 5.2 guarantees that the input ontologies are closed by classification, and therefore from Proposition 5.1, the induced subgraph obtained using only (named) concepts from a single input ontology, is a sound and complete encoding for the subsumption relation of such ontology. Given that all the elements of the (sub)path $[C_1, \ldots, C_n]$ belong to the same input ontology, $\mathcal{O}_j \models C_1 \sqsubseteq C_n$ holds, and this concludes the proof.

□

We are now ready to explain the potential source of incompleteness for our direct violation detection algorithm exclusively using the structural index.

Graph-based Direct Violation Detection. The structural index directly encodes the classification DAG computed by an OWL 2 reasoner. Unfortunately, during the graph reduction phase, in order to ensure minimality over the computed DAG, reasoners are not forced to keep asserted

axioms, thus altering the direct subconcept relation.[4] This is problematic to our aim, given that, using the structural index of the aligned ontology, we may loose some direct violations, due to the "absence" of some entities in the set of direct subconcepts. Due to this technical reason, the aforementioned technique is sound but not complete , and needs to be complemented with the, already introduced, additional steps of lines 9–14. To this aim, we again employ the graph representation, introduced in Chapter 5 in Definition 5.2, encoding both asserted and inferred axioms, thus overcoming the limitation highlighted above.

At this point, the algorithm computes all subsumption violations $allSubViols$ (line 10). Given that, as discussed above, the direct violation detection algorithm is sound but possibly incomplete, additional direct violations may exist in the set difference between $allSubViols$ and $viols$. This set difference (candidate direct violations) is then passed to the graphDirViolCheck function (line 12), that tries to find an appropriate path by means of a *depth-first seach* (DFS) or *breadth-first seach* (BFS) visit over (a subgraph of) the graph representation of the aligned ontology. This function returns the set of additional direct violations $dirViols$, that is added to the set of violations to return (line 13).

We now describe in detail the graphDirViolCheck function (shown in Algorithm 21), that takes as input the aligned ontology $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$, its structural index $SI_{\mathcal{U}}$, the set of candidate direct violations $violsChk$, and a Boolean flag $useDFS$ for discriminating between the use of a DFS or BFS graph visit.

Firstly, using the structural index, we isolate the candidate violations for the currently analyzed input ontology (line 3). Then, we extract the seed signature (line 4) in a way such that the corresponding locality module ensures completeness for the graph-based detection technique, and we contextually compute its graph representation (line 5).

The module is employed to reduce the size of the search space, as well as the memory occupation of the graph representation. Given that we require to build a graph including all the asserted and inferred subsumption axioms, when the input ontologies and the alignment between them are of significant size, the memory occupation could be critical.

Proposition 6.9 states and proves completeness for the graph-based direct violation detection technique described above.

Proposition 6.9. The use of $\bot$-locality modules in the graph-based direct violation detection technique does not affect its completeness.

Proof. The proof directly follows from observing that Proposition 6.6 and 6.7 can be seen, respectively, as the base step and the inductive one for a recursive "reachability" algorithm by subsumption relation. Proposition 6.6 tells us that, given any concept in the seed signature, all its subsumptions (having the considered concept as subsumee), can be entailed. Proposition 6.7

[4] Of course the subsumption relation is preserved, and the reduction is semantically safe.

Algorithm 21 graphDirViolCheck function for graph-based direct violations detection

Input: $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2}$: aligned ontology; $SI_{\mathcal{U}}$: structural index of the aligned ontology; $violsChk$: violations to check; $useDFS$: Boolean flag
Output: $viols$: set of confirmed direct violations

```
1: viols ← ∅
2: for each i in {1,2} do                                   ▷ Process the violations of the input ontology separately
3:     violsChk_i ← {A ⊑ B | A ⊑ B ∈ violsChk and SI_U.getOntoId(A) = i}
4:     seed ← {A | A ⊑ B ∈ violsChk_i}
5:     G ← G(moduleExtractor(O^M_{O1,O2}, seed))
6:     for each v ← A ⊑ B in violsChk_i do
7:         visited ← ∅
8:         toProcess ← [A]                                  ▷ Initialize toProcess queue
9:         while not toProcess.isEmpty() do
10:            C ← toProcess.poll()
11:            if visited.contains(C) then
12:                continue
13:            end if
14:            visited.add(C);
15:            for each (C,C',_) in G.getOutArcs(C) do
16:                if C' = B then
17:                    viols.add(v)
18:                    goto line 6                          ▷ Process next violation
19:                end if
20:                if SI_U.getOntoId(C') = i then           ▷ Path cannot traverse concepts from the same ontology of the violation
21:                    continue
22:                end if
23:                if useDFS then
24:                    toProcess.addFirst(C')
25:                else
26:                    toProcess.addLast(C')
27:                end if
28:            end for
29:        end while
30:    end for
31: end for
32: return viols
```

is analogous, but for the subsumer of the axiom. By repeatedly applying Proposition 6.7, we have that any "step" of a subsumption "chain" starting at an element of the seed signature, is still derivable in the extracted module, thus proving the proposition.

□

The final step of our algorithm for detecting and repair conservativity violations, is composed by a mapping repair step.

Mapping Repair. The step 13 of Algorithm 16 uses the mapping (incoherence) repair algorithm of LogMap, for the extended Horn propositional formulas ${\mathcal{P}_1}^d$ and ${\mathcal{P}_2}^d$, and the input mappings $\mathcal{M}$. The algorithm computes a repair, as already described in Section 6.3.2.

Example 6.8 provides an example of mapping repair for solving a conservativity violation.

Example 6.8. Consider the propositional encoding $\mathcal{P}_1$ and $\mathcal{P}_2$ of the axioms of Table 6.1 and the mappings $\mathcal{M}$ in Table 6.2, seen as propositional rules. ${\mathcal{P}_1}^d$ and ${\mathcal{P}_2}^d$ have been created by adding disjointness rules to $\mathcal{P}_1$ and $\mathcal{P}_2$, according to Algorithm 17 or 18. For example, ${\mathcal{P}_2}^d$ includes the rule $\psi = \mathcal{O}_2{:}Well \wedge \mathcal{O}_2{:}Borehole \rightarrow \mathsf{false}$. The mapping repair algorithm identifies the propositional theory ${\mathcal{P}_1}^d \cup {\mathcal{P}_2}^d \cup \mathcal{M} \cup \{\mathsf{true} \rightarrow \mathcal{O}_1{:}ExplorationWellbore\}$ as unsatisfiable. This is due to the combination of the mappings m_3 and m_4, the propositional projection of axioms β_1 and β_2, and the rule ψ. The mapping repair algorithm also identifies m_3 and m_4 as the cause of the unsatisfiability, and discards m_3, since its confidence is lower than that of m_4 (see Table 6.2). $\Diamond$

Finally, Algorithm 16 returns as output the number of added disjointness rules during the process $disj$, a set of mappings $\mathcal{M}'$, and an (approximate) repair $\mathcal{R}^{\approx}$ such that $\mathcal{M}' = \mathcal{M} \setminus \mathcal{R}^{\approx}$. $\mathcal{M}'$ is coherent with respect to ${\mathcal{P}_1}^d$ and ${\mathcal{P}_2}^d$ (according to the propositional case of Definition 4.24). Furthermore, the propositional theory $\mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{M}'$ does not contain any conservativity principle violation with respect to $\mathcal{P}_1$ and $\mathcal{P}_2$ (according to the propositional case of Definition 4.26). However, our encoding is incomplete, and we cannot guarantee that $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}'$ does not contain conservativity principle violations with respect to $\mathcal{O}_1$ and $\mathcal{O}_2$. Nonetheless, our evaluation suggests that the number of remaining violations after repair is typically small (see Section 6.5).

6.5 Experimental Evaluation

In this section we evaluate the feasibility of using our method to detect and correct conservativity principle violations in practice. Section 6.5.1 provides the necessary details on the implementation. Section 6.5.2 compares the two proposed methods for enriching the Horn projections of the input ontologies with disjointness clauses, using the ontologies and reference alignments of *OAEI* 2013. Section 6.5.3, instead, analyzes the behaviour of our repair algorithm on an extended dataset (*i.e.*, alignments computed by matching systems participating at the *OAEI* 2012–2014). Section 6.5.4 analyzes the impact of conservativity repair on the alignment quality in terms of precision, recall, and f-measure. Finally, Section 6.5.5 shows the GUI for supporting manual debugging of subsumption violations.

6.5.1 Implementation Details

In this section we discuss the relevant details concerning our implementation. The prototype is written in *Java*. *OWL-API* 3.5.1 is used for ontologies manipulation, including classification by means of *HermiT* 1.3.8 reasoner. Given the computational hardness of computing classification on the aligned ontologies for some tasks, we rely on a 180 seconds timeout, after which the classification is computed using *ELK* 0.4.1 reasoner [KKS11]. In this way, a lower bound on the

Algorithm 22 Conducted evaluation for the SubRepair algorithm

Input: $\mathcal{O}_1$, $\mathcal{O}_2$: input ontologies $\mathcal{M}$: alignment between $\mathcal{O}_1$ and $\mathcal{O}_2$

1: $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \leftarrow \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$
2: Store size of $\mathsf{Sig}(\mathcal{O}_1)$ (**I**), $\mathsf{Sig}(\mathcal{O}_2)$ (**II**) and $\mathcal{M}$ (**III**)
Compute number of conservativity principle violations:
3: $\mathsf{basicViol} \leftarrow |\mathsf{basicViol}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})| + |\mathsf{basicViol}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})|$ (**IV**) ▷ basic violations, Definition 4.26, Section 4.3.6
4: $\mathsf{diff}^{\approx} \leftarrow |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_1)}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})| + |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_2)}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})|$ (**V**) ▷ general notion, Definition 4.23, Section 4.3.4
5: Compute two repairs $\mathcal{R}^{\approx}$ using Algorithm 16 for $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{M}$, with $optimization$ flag set to false (see Table 6.6) and true (see Table 6.7)
6: Store number of added disjointness $disj$ (**VI** and **XII**), size of repair $|\mathcal{R}^{\approx}|$ (**VII** and **XIII**), time to compute disjointness rules t_d (**VIII** and **XIV**), and time to compute the mapping repair t_r (**IX** and **XV**)
7: $\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \leftarrow \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M} \setminus \mathcal{R}^{\approx}$
Compute number of remaining conservativity principle violations:
8: $\mathsf{basicViol} \leftarrow |\mathsf{basicViol}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})| + |\mathsf{basicViol}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})|$ (**X** and **XVI**)
9: $\mathsf{diff}^{\approx} \leftarrow |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_1)}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})| + |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_2)}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2})|$ (**XI** and **XVII**)

number of conservativity principle violations can be provided.[5] In particular, for the aligned ontologies involving SNOMED, it is well known that no OWL 2 reasoner succeeds in classification task in reasonable time [JRCH12].

As already discussed, the structural indexing and repair facilities of LogMap have been employed, with some extensions (*e.g.*, structural index update support for disjointness clause addition). LogMap is used also to load and store the alignments.

The test environment consists of a desktop computer equipped with 32*GB DDR*3 *RAM* at 1333*MHz*, and an *AMD Fusion FX 4350* (quad-core, each running at 4.2*GHz*) as *CPU*. The dataset is stored on a 128*GB SSD*, where the operating system (*Ubuntu* 12.04, 64-bit version) is also installed.

All the results involving time measurements are averaged over multiple repetitions, in order to increase the statistical significance of the experimental results.

6.5.2 Disjointness Addition

The evaluation shown in Algorithm 22 aims at comparing the efficiency and effectiveness of our alternative methods (namely basic and optimized) for enriching Horn projections of the input ontologies with disjointness clauses. The Roman numbers refer to measurements that are stored during the evaluation. This enrichment phase is crucial for the reduction of the conservativity violations repair problem to a consistency repair. Given the prohibitive runtime of the basic method, we have restricted the evalutation to the ontologies and reference mapping sets of the OAEI 2013 campaign:[6]

(i) largebio: this dataset includes the biomedical ontologies FMA, NCI and (a fragment of)

[5] When an approximated result is shown it is always preceeded by the "$>$" symbol.

[6] More information about the used ontology versions can be found at `http://oaei.ontologymatching.org/2013/`

SNOMED, and reference mappings based on the UMLS [Bod04].

(ii) *anatomy*: the *anatomy* dataset involves the Adult Mouse Anatomy (MO) ontology and a fragment of the NCI ontology (NCI_{Anat}), describing human anatomy. The reference alignment has been manually curated [ZMB04].

(iii) *library*: this *OAEI* track includes the real-word thesauri STW and TheSoz from the economic and social sciences. The reference mappings have been manually validated.

(iv) *conference*: this track uses a collection of 16 ontologies from the domain of academic conferences [vSB$^+$05]. Currently, there are 21 manually created mapping sets among 7 of the ontologies.

Dataset	$\mathcal{O}_1 \sim \mathcal{O}_2$	**Problem size**			**Original Violations**	
		I	**II**	**III**	**IV**	**V**
		$\lvert\mathsf{Sig}(\mathcal{O}_1)\rvert$	$\lvert\mathsf{Sig}(\mathcal{O}_2)\rvert$	$\lvert\mathcal{M}\rvert$	**basicViol**	**diff$^{\approx}$**
LargeBio	**SNOMED∼NCI**	122,519	66,914	36,405	>606,220	>628,858
	FMA∼SNOMED	79,042	122,519	17,212	125,232	127,668
	FMA∼NCI	79,042	66,914	5,821	19,740	19,799
Anatomy	**MO∼NCI_{Anat}**	2,747	3,306	3,032	1,321	1,335
Library	**STW∼TheSoz**	6,575	8,376	6,322	42,045	42,872
Conference	**cmt∼confof**	89	75	32	11	11
	conference∼edas	124	154	34	8	8
	conference∼iasted	124	182	28	9	9
	confof∼ekaw	75	107	40	6	6
	edas∼iasted	154	182	38	6	6

Table 6.5: Test cases and violations with original reference mappings.

Dataset	$\mathcal{O}_1 \sim \mathcal{O}_2$	**Solution size**		**Times**		**Remaining Violations**	
		VI	**VII**	**VIII**	**IX**	**X**	**XI**
		#disj	$\lvert\mathcal{R}^{\approx}\rvert$	$\mathbf{t}_d$**(s)**	$\mathbf{t}_r$**(s)**	**basicViol**	**diff$^{\approx}$**
LargeBio	**SNOMED∼NCI**	–	–	–	–	–	–
	FMA∼SNOMED	1,124,541	5,671	6,836	285	0	201
	FMA∼NCI	403,198	1,514	684	26	102	112
Anatomy	**MO∼NCI_{Anat}**	1,334,638	306	177	44	0	3
Library	**STW∼TheSoz**	591,054	2,221	1,005	85	0	38
Conference	**cmt∼confof**	385	3	0.01	0.01	0	0
	conference∼edas	1,924	3	0.02	0.01	0	0
	conference∼iasted	2,212	3	0.03	0.01	0	0
	confof∼ekaw	824	6	0.01	0.01	0	0
	edas∼iasted	2,315	4	0.03	0.01	1	1

Table 6.6: Results of our basic method to detect and solve conservativity principle violations.

Table 6.5 shows the size of the evaluated ontologies and mappings (**I**, **II**, and **III**). For the *conference* track we have selected only 5 pair of ontologies for which the reference mappings lead to more than five conservativity principle violations. Equivalence mappings are counted as two

Dataset	$\mathcal{O}_1 \sim \mathcal{O}_2$	Solution size		Times		Remaining Violations	
		XII	XIII	XIV	XV	XVI	XVII
		#disj	$\lvert\mathcal{R}^{\approx}\rvert$	t_d(s)	t_r(s)	basicViol	diff$^{\approx}$
LargeBio	SNOMED∼NCI	606,220	13,856	60	687	>436	>1,867
	FMA∼SNOMED	125,232	5,565	14	45	0	176
	FMA∼NCI	19,740	1,485	16	2.48	102	111
Anatomy	MO∼NCI$_{\text{Anat}}$	1,321	317	0.74	0.38	0	3
Library	STW∼TheSoz	42,045	2,183	2.92	8.84	0	38
Conference	cmt∼confof	11	3	0.02	0.01	0	0
	conference∼edas	8	3	0.02	0.01	0	0
	conference∼iasted	9	1	0.13	0.01	0	0
	confof∼ekaw	6	5	0.02	0.01	0	0
	edas∼iasted	6	4	0.12	0.01	1	1

Table 6.7: Results of our optimized method to detect and solve conservativity principle violations.

subsumption mappings, and hence $\mathcal{M}$ contains subsumption mappings only (*i.e.*, equivalence mappings are split). Table 6.5 also shows the conservativity principle violations for the reference mappings (**IV** and **V**). For *largebio* and *library* tracks the number is expecially large, using both our variant and the general notion of the conservativity principle.

Tables 6.6 and 6.7 show the obtained results for our method using the basic and optimized algorithms to add disjointness axioms, respectively.

We have run the experiments allocating 12 GB of RAM for the JVM. The obtained results are summarized as follows:

(i) The number of added disjointness rules $\#disj$ (**VI**), as expected, is very large in the basic algorithm and the required time t_d to compute them and update the index (**VIII**) is prohibitive when involving SNOMED (almost 10 hours for FMA-SNOMED and it did not finish for SNOMED-NCI). This is clearly solved in our optimized algorithm that considerably reduces the number of necessary disjointness rules (**XII**), and which requires only 60 seconds to compute them and update the index in the SNOMED-NCI case (**XIV**).

(ii) The computed repairs $\mathcal{R}^{\approx}$ (**VII** and **XIII**) using both the basic and optimized algorithms are of comparable size. This suggests that the large number of added disjointness clauses in the basic algorithm does not have a negative impact (in terms of aggressiveness) on the repair process.

(iii) Repair times t_r (**IX** and **XV**) are small and they do not represent a bottleneck in spite of the large number of added disjointness rules.

(iv) The conservativity principle violations using both algorithms and considering our variant (**X** and **XVI**) are completely removed in the *anatomy* and *library* cases, and almost completely removed in the *conference* and *largebio* tracks.

(v) The number of missed violations is only slightly higher when considering the general notion of the conservativity principle (**XI** and **XVII**), which suggests that our (approximate) variant is also suitable in practice. Furthermore, in several test cases these violations are also almost removed.

(vi) The computed repairs $\mathcal{R}^{\approx}$, using both algorithms (**VII** and **XIII**), can be rather aggressive, removing from around 10% of the input aligment (*anatomy* and *conference* tracks) to more than 30% (*largebio* and *library* tracks).

In summary, the results suggest that the effectiveness of the two methods is comparable. However, the optimized variant proved to be more efficient than the basic one. For this reason, in all the further experiments we will exclusively consider the optimized variant.

6.5.3 Analysis of Ontology Matchers Alignments

For testing the effect of our repair algorithm, we repeated the evaluation of Algorithm 22 for the alignments computed by the systems participating at the *OAEI* (years 2012–2014), in the *anatomy*, *largebio*, *conference*, and *library* tracks.

For this experiment, the amount of RAM allocated for the Heap of the JVM is $26GB$, in order to minimize the running time of the Garbage Collector. All the tests for *conference*, *anatomy* tracks were able to run with at least $4GB$. Most of the tests for *largebio* and *library* tracks could run with $8GB$ of RAM, with few outliers requiring more memory, up to 2 tasks of *library* requiring a higher amount of memory (at least $11GB$ for AML 2013 and $13GB$ for Hertuda 2013) to be executed. Despite the allocation of $26GB$, two tasks of *library* (Aroma 2012 and XMapGen 2013) were not able to complete. However, these tasks are evidently degenerate cases, given the extraordinary number of detected violations (more than $37M$ basic and $8M$ equivalence violations for Aroma 2012, and more than $55M$ basic and $17M$ equivalence violations for XMapGen 2013), specifically if compared with the violations affecting the reference alignment for *library* track (see Table 6.5).

The amount of required memory is highly influenced by the number of detected violations, that are stored during our experiments in order to be able to compare them with the set of unsolved violations, after the application of the repair step(s). In order to be able to accurately measure the detection time, this phase is performed three times, one for each violation kind, and the results are stored separately. Another source of memory occupation is represented by the number of disjointness clauses that are added to the graph encoding of the Horn projections, while running the D&G algorithm. We remind that, for each disjointness clause, two arcs need to be added to the graph. Of course the amount of memory strictly required by our approach is lower, because it is only needed to store basic violations, that can be dropped after their addition to the Horn projections (*i.e.*, before the D&G execution).

It is well known that the Java language is not optimized w.r.t. memory consumption.[7] For instance, in the case of XMapGen, more than $12GB$ are used to simply store the detected violations (as pair of Integer values representing identifiers of the structural index) in an *ArrayList* data structure. Using a programming language allowing for more advanced memory manipulation (*e.g.*, C or C++) could help to significantly reduce the memory needed by our algorithms.

Tables 6.8–6.11 summarize relevant features of the dataset and analysis results. All the results computed after a grouping operation present the format *mean* (*standard deviation*) in place of a single aggregate value. The format of Tables 6.8 and 6.10, describing the measures related to the problem size, coincides with that of Table 6.5. The format of Tables 6.9 and 6.11, instead, coincides with that of Table 6.7 (the table describing the results for the optimized variant, the only one employed here).[8]

However, differently from Section 6.5.2, we do not consider reference alignments but alignments computed by different systems in the context of the *OAEI* campaign for the years 2012–2014. As already discussed, the *OAEI* campaign is composed by different tracks, in turn possibly composed by multiple tasks (*e.g.*, *largebio* track has 6 tasks, while conference has 21 tasks). The list of participating systems is subject to change each year, and despite all the systems are runned against each track (and task), some of them may not be able to produce a result for some track or task. For this reason, given that we present averaged results aggregating by system or track, it is interesting to know how many alignments are used to compute each provided result. This information is proposed in Tables 6.8 and 6.10, in the column $\#\mathcal{M}$, that therefore corresponds to the number of alignments used to compute the aggregate result for the considered row.

Tables 6.8 and 6.10 confirm that our conservativity violation notion is a good approximation of the general notion in practice (the difference between columns **IV** and **V**, in both tables, is limited).

From Tables 6.9 and 6.11 we can see that our repair algorithm is effective, because the quantity of unsolved violations is extremely reduced (considering both notions, **XVI** and **XVII**).

(i) Despite the number of violations affecting alignments computed by matchers is on average higher than those affecting the reference alignments, the scalability of the detection algorithm and structural index update is evident, with on average 77 seconds in the worst case of LogMap2Noe (**XIV**, Table 6.9). Table 6.11 shows that the required time t_d is directly influenced by the size of the input ontologies, more than from the number of violations. However, the size of the input ontologies is not conclusive. For instance, despite the size of

[7] In most of the JVM implementations, a word is required to store each variable, including Boolean primitive datatype ones, and at least 2 words for each allocated object, even if having no fields, like for objects of type *Object*. For further details refer to `http://kohlerm.blogspot.it/2008/12/how-much-memory-is-used-by-my-java.html`.

[8] The gap in the roman numbers is therefore due to the missing results for the basic variant, motivated at the end of Section 6.5.2.

	Problem Size			Original Violations		Alignments
	I	II	III	IV	V	
	$\lvert\mathsf{Sig}(\mathcal{O}_1)\rvert$	$\lvert\mathsf{Sig}(\mathcal{O}_2)\rvert$	$\lvert\mathcal{M}\rvert$	basicViol	$\mathsf{diff}^{\approx}$	$\#\mathcal{M}$
aml	12,224(29,889)	10,806(27,190)	3,462(7,054)	327,899(2,030,773)	331,163(2,042,411)	59
amlbk	12,527(30,946)	10,936(28,172)	3,235(7,146)	46,468(120,759)	47,797(124,906)	28
amlbkr	57,614(45,446)	50,061(44,114)	13,944(8,532)	201,763(220,499)	207,974(229,734)	6
amlbku	57,614(45,446)	50,061(44,114)	19,944(13,448)	394,605(308,040)	403,076(314,020)	6
amlbkur	57,614(45,446)	50,061(44,114)	18,907(12,874)	486,519(455,149)	499,143(470,545)	6
amlr	57,614(45,446)	50,061(44,114)	13,359(8,405)	185,132(220,000)	190,467(228,646)	6
aot	391(933)	560(1,501)	692(1,872)	2,512(7,807)	2,533(7,867)	22
aotl	382(912)	539(1,469)	110(327)	430(1,486)	432(1,490)	23
aroma	5,540(14,285)	7,336(24,043)	2,042(5,072)	41,521(114,796)	42,356(117,442)	27
ase	107(27)	123(44)	47(27)	64(136)	66(137)	20
autom	274(770)	413(1,372)	179(768)	282(1,321)	283(1,327)	22
cidercl	237(576)	270(697)	208(737)	2,274(10,284)	2,288(10,346)	21
codi	506(1,433)	618(1,817)	419(1,445)	12,514(59,481)	12,750(60,612)	23
cromatcher	107(36)	121(46)	81(22)	26(38)	27(38)	15
gomma	17,023(33,634)	15,225(31,294)	4,576(7,354)	547,306(3,129,072)	550,954(3,144,329)	37
gommabk	38,540(42,323)	35,059(41,568)	16,185(11,274)	181,706(191,482)	191,265(202,308)	7
gommasbk	57,614(45,446)	50,061(44,114)	18,493(10,722)	237,553(195,121)	248,350(204,851)	6
hertuda	1,021(2,411)	1,368(3,245)	1,205(3,056)	1,070,035(5,279,337)	1,074,438(5,300,895)	50
hotmatch	1,021(2,411)	1,368(3,245)	661(1,562)	4,356(18,533)	4,442(18,921)	50
iama	12,321(30,409)	10,848(27,669)	1,722(4,455)	11,534(34,419)	11,796(35,326)	29
logmap	11,362(28,458)	10,158(26,202)	3,332(6,999)	65,509(179,944)	67,388(185,803)	88
logmap2noe	44,505(43,018)	40,351(42,874)	14,837(9,642)	262,095(311,406)	269,874(322,187)	6
logmapbio	49,776(46,381)	43,382(43,977)	13,713(9,794)	358,123(410,317)	367,913(425,000)	7
logmapbk	57,614(45,446)	50,061(44,114)	14,654(9,536)	313,796(342,009)	321,864(353,585)	6
logmapc	12,321(30,409)	10,848(27,669)	2,126(4,523)	403(1,834)	478(2,052)	29
logmaplt	11,417(28,618)	10,178(26,353)	2,662(6,201)	338,820(1,671,623)	342,082(1,684,896)	87
maasmatch	607(1,629)	831(2,281)	917(2,622)	157,203(1,223,543)	158,621(1,234,257)	71
mapsss	2,258(8,392)	1,832(5,379)	867(3,004)	12,953(65,926)	13,300(67,809)	49
medley	110(31)	121(44)	98(43)	92(185)	93(185)	21
odgoms	870(2,306)	1,182(3,133)	777(2,295)	12,249(63,650)	12,413(64,510)	48
omreasoner	5,857(18,402)	4,539(14,089)	927(2,929)	4,876(15,362)	4,965(15,714)	25
ontok2	110(31)	121(44)	18(6.73)	0.29(0.9)	0.29(0.9)	21
optima	506(1,433)	618(1,817)	179(500)	4,259(18,636)	4,340(18,994)	23
rimom	110(31)	121(44)	114(48)	134(105)	139(108)	21
rsdlwb	640(1,547)	866(2,150)	168(473)	76(213)	77(215)	24
servomap	10,965(27,944)	9,844(25,912)	3,215(6,862)	148,063(565,523)	150,113(571,146)	58
servomapl	9,609(25,712)	8,839(24,477)	3,233(6,898)	122,405(358,776)	124,813(364,640)	29
sphere	57,614(45,446)	50,061(44,114)	9,331(7,650)	102,947(128,423)	105,714(132,577)	6
stringsauto	640(1,547)	866(2,150)	358(968)	752(2,223)	759(2,241)	24
synthesis	110(31)	121(44)	17(6.79)	0.33(0.91)	0.33(0.91)	21
toast	2,747(0)	3,306(0)	2,678(0)	2,597(0)	2,612(0)	1
wesee	371(1,094)	446(1,380)	245(974)	2,977(18,757)	3,043(19,186)	45
wmatch	308(757)	405(1,149)	258(1,037)	384(2,324)	390(2,356)	45
xmap	12,321(30,409)	10,848(27,669)	3,852(7,909)	902,978(4,521,677)	908,611(4,541,750)	29
xmapgen	473(1,620)	630(2,221)	242(811)	1,328(4,899)	1,340(4,941)	45
xmapsig	605(1,838)	799(2,475)	267(823)	3,001(16,040)	3,093(16,613)	46
yam++	10,965(27,944)	9,844(25,912)	3,375(6,945)	133,905(476,507)	136,107(482,130)	58

Table 6.8: Measures related to problem size for *OAEI* 2012-2014 dataset, grouped by matcher.

	Solution Size		Times		Remaining Violations	
	XII	XIII	XIV	XV	XVI	XVII
	$\#\mathbf{disj}$	$\lvert\mathcal{R}^{\approx}\rvert$	$\mathbf{t}_d$(s)	$\mathbf{t}_r$(s)	basicViol	$\mathbf{diff}^{\approx}$
aml	327,899(2,030,773)	884(1,906)	11(26)	142(687)	7.36(27)	46(126)
amlbk	46,468(120,759)	662(1,637)	9.4(24)	48(134)	6.39(24)	36(126)
amlbkr	201,763(220,499)	3,164(2,549)	35(24)	182(188)	31(47)	154(216)
amlbku	394,605(308,040)	6,091(4,488)	50(45)	374(396)	46(71)	502(545)
amlbkur	486,519(455,149)	6,514(5,124)	61(79)	611(701)	145(210)	655(626)
amlr	185,132(220,000)	2,923(2,403)	33(22)	172(184)	20(41)	115(161)
aot	2,512(7,807)	167(522)	1.48(5.29)	0.34(1.15)	59(275)	61(281)
aotl	430(1,486)	18(47)	1(3.35)	0.04(0.14)	0.52(2.5)	0.61(2.71)
aroma	41,521(114,796)	428(1,137)	8.51(26)	36(127)	4.11(20)	15(46)
ase	64(136)	7.85(9.29)	0.11(0.31)	0	0.15(0.49)	0.75(1.12)
autom	282(1,321)	22(104)	2.45(11)	0.05(0.23)	1.73(7.88)	1.73(7.88)
cidercl	2,274(10,284)	43(177)	0.22(0.54)	0.19(0.86)	0.52(1.66)	0.67(1.74)
codi	12,514(59,481)	95(410)	0.43(1.71)	2.36(11)	0.04(0.21)	0.22(0.6)
cromatcher	26(38)	8.41(6.59)	0.15(0.36)	0	0.13(0.52)	0.13(0.52)
gomma	547,306(3,129,072)	975(1,769)	20(41)	221(1,092)	10(32)	48(90)
gommabk	181,706(191,482)	3,638(2,698)	59(61)	163(210)	58(71)	946(1,610)
gommasbk	237,553(195,121)	4,177(2,445)	38(30)	213(225)	54(77)	1,103(1,760)
hertuda	1,070,035(5,279,337)	616(1,719)	11(49)	490(2,406)	4.08(20)	10(31)
hotmatch	4,356(18,533)	93(276)	1.34(3.96)	1.06(4.06)	4.44(22)	4.76(22)
iama	11,534(34,419)	263(757)	6.33(16)	17(52)	1.34(6.86)	18(60)
logmap	65,509(179,944)	906(2,047)	11(28)	70(229)	18(73)	67(232)
logmap2noe	262,095(311,406)	3,916(3,082)	77(67)	302(369)	97(135)	298(414)
logmapbio	358,123(410,317)	4,008(3,268)	43(45)	396(536)	85(177)	359(504)
logmapbk	313,796(342,009)	4,034(3,216)	43(37)	382(487)	86(159)	321(488)
logmapc	403(1,834)	23(89)	4.88(12)	8.22(25)	5.14(21)	47(128)
logmaplt	338,820(1,671,623)	685(1,648)	13(30)	136(585)	5.37(22)	28(72)
maasmatch	157,203(1,223,543)	208(781)	4.2(14)	25(184)	2.69(13)	4.1(17)
mapsss	12,953(65,926)	142(582)	4.96(24)	6.68(40)	5.84(35)	7.98(38)
medley	92(185)	14(14)	0.34(0.75)	0	0.24(0.89)	0.24(0.89)
odgoms	12,249(63,650)	164(613)	1.54(4.56)	3.15(16)	4.29(20)	6.46(23)
omreasoner	4,876(15,362)	144(465)	2.51(5.91)	4.46(13)	1.52(7.39)	2.44(8.54)
ontok2	0.29(0.9)	0.14(0.48)	0.04(0.05)	0	0.05(0.22)	0.05(0.22)
optima	4,259(18,636)	40(121)	0.26(0.76)	0.59(2.67)	0.04(0.21)	0.26(1.05)
rimom	134(105)	49(39)	0.84(2.92)	0.19(0.62)	0.05(0.22)	0.05(0.22)
rsdlwb	76(213)	11(33)	0.76(3.19)	0.06(0.18)	0.04(0.2)	0.12(0.45)
servomap	148,063(565,523)	874(2,017)	18(48)	125(363)	5.9(23)	34(101)
servomapl	122,405(358,776)	718(1,549)	19(46)	83(238)	12(32)	45(126)
sphere	102,947(128,423)	1,566(1,588)	32(23)	113(121)	18(41)	229(248)
stringsauto	752(2,223)	45(123)	0.92(3.72)	0.15(0.47)	0.17(0.56)	0.29(0.69)
synthesis	0.33(0.91)	0.24(0.62)	0.04(0.05)	0	0.05(0.22)	0.05(0.22)
toast	2,597(0)	311(0)	0.63(0)	0.32(0)	0	0
wesee	2,977(18,757)	46(243)	0.19(0.76)	0.67(4.27)	0.04(0.21)	0.29(1.38)
wmatch	384(2,324)	43(210)	1.52(9.75)	0.06(0.3)	2.47(16)	2.84(18)
xmap	902,978(4,521,677)	974(2,142)	20(53)	470(2,216)	4.38(19)	31(86)
xmapgen	1,328(4,899)	46(163)	0.7(3)	0.19(0.95)	3.4(23)	3.76(22)
xmapsig	3,001(16,040)	52(175)	0.73(2.87)	0.59(2.97)	3.33(22)	4.17(23)
yam++	133,905(476,507)	820(1,748)	14(35)	79(214)	6.4(24)	31(78)

Table 6.9: Measures related to solution size for *OAEI* 2012-2014 dataset, grouped by matcher.

	Problem Size			Original Violations		Alignments
	I	II	III	IV	V	
	$\lvert\mathsf{Sig}(\mathcal{O}_1)\rvert$	$\lvert\mathsf{Sig}(\mathcal{O}_2)\rvert$	$\lvert\mathcal{M}\rvert$	basicViol	diff$^{\approx}$	$\#\mathcal{M}$
anatomy	2,747(0)	3,306(0)	2,607(816)	6,562(12,844)	6,619(12,911)	47
conference	110(30)	121(43)	33(35)	17(86)	17(87)	1,104
largebio_big	18,340(19,857)	13,450(6,995)	11,789(8,729)	154,566(237,174)	159,711(245,746)	122
largebio_small	84,567(30,642)	79,217(33,050)	13,868(9,056)	270,491(268,151)	276,684(274,228)	91
library	6,575(0)	8,376(0)	6,363(3,168)	5,121,627(8,375,902)	5,153,135(8,413,450)	32

Table 6.10: Measures related to problem size for *OAEI* 2012-2014 dataset, grouped by track.

	Solution Size		Times		Remaining Violations	
	XII	XIII	XIV	XV	XVI	XVII
	#disj	$\lvert\mathcal{R}^{\approx}\rvert$	t_d(s)	t_r(s)	basicViol	diff$^{\approx}$
anatomy	6,562(12,844)	396(413)	0.77(0.4)	1.22(2.6)	0	2.23(1.99)
conference	17(86)	3.4(10)	0.17(1.45)	0.01(0.09)	0.15(1.33)	0.18(1.34)
largebio_big	154,566(237,174)	2,897(2,847)	25(18)	134(286)	79(151)	234(348)
largebio_small	270,491(268,151)	3,105(2,573)	76(56)	343(353)	18(43)	267(690)
library	5,121,627(8,375,902)	2,865(2,196)	48(75)	1,966(3,690)	0	67(54)

Table 6.11: Measures related to solution size for *OAEI* 2012-2014 dataset, grouped by track.

the input ontologies of the *largebio-big* is higher than that of the *largebio-small* track, the required time is lower. The motivation is that, on average, the alignments for the *largebio-small* track are of higher size, and this leads to bigger aligned ontologies (due to the use of modularization techniques that select only the relevant part of the input ontologies w.r.t. the given alignment).

(ii) Repair times t_r (**XV**) are small and they do not represent a bottleneck in spite of the large number of added disjointness rules. An exception is represented by the *library* track, where an average repair time of 33 minutes is required. However, the runtime is acceptable, considering the impressive average number of violations (*i.e.*, $5M$), to be added to the graph encoding of the Horn projection of the aligned ontology.

(iii) The conservativity principle violations, considering our variant (**XVI**), are completely removed in the *anatomy* and *library* cases, and almost completely removed in the *conference* and *largebio* tracks (with an average of 0.05% unsolved violations in the worst case of the *largebio-big* track).

(iv) The number of missed violations is only slightly higher when considering the general notion of the conservativity principle (**XVII**), which suggests that our (approximate) variant is also suitable in practice. Furthermore, the number of unsolved violations using this notion is negligible (an average of 0.15% in the worst case represented by the *largebio-big* track).

(v) The computed repairs $\mathcal{R}^{\approx}$ (**XIII**) can be aggressive, given that average removal ranges from 10% of the original alignment for the *conference* track, up to an average of 45% for the *library* track. From Table 6.11 the expected positive correlation between the repair size and the number of detected violations clearly emerges.

This extended set of experiments basically confirms the effectiveness and efficiency of the detection and repair algorithms. This evaluation shows also that the violation detection algorithm is more influenced by the size of the involved ontologies and alignments, while the time required by the repair algorithm is strongly correlated with the number of violations to repair.

Different groupings for the same experimental values are provided in [Sol15], Section 2.1.

6.5.4 Repair Effects on Alignment Quality

In Figure 6.11 the average impact on precision, recall and f-measure due to the application of the computed repairs is shown, analogously to what reported in Section 5.5.5 for equivalence violations repair.

Again, the results are grouped by track (that is, *anatomy*, *conference*, *largebio* and *library*), and the two variants of *largebio* track (that is, *largebio-small* and *largebio-big*) are separately analyzed.

The impact of alignment repair is computed as the percentual of gain (that corresponds to a loss for negative values) for each measure computed for a repaired alignment, compared to the same measure computed for the original alignment. In both cases the measures are computed w.r.t. the reference alignment, provided by the *OAEI* organizers, but differently from Section 5.5.5, the original computed alignment is the output of a (full) consistency repair. This is required to remove any bias possibly introduced by this propedeutic and necessary repair step.[9]

Given that we are investigating the effect of alignment repair, we filtered any alignment without conservativity violations, because the empty repair always implies a void gain.

Figure 6.11 confirms the general trends for precision and recall for a repair algorithm (increase and decrease, respectively). The gains for precision and recall result in a slight increase of the resulting f-measure for *conference* track, and a slight decrease for *anatomy* track. Instead, for both variants of *largebio* and *library* track, the loss is higher, and lies in the range $[0, 10]\%$.

The results show that the effect of the subsumption violations repair algorithm, in terms of performance w.r.t. a reference alignment, is higher than in case of equivalence violations (Section 5.5.5). By a comparison between the results for the two repair algorithms, a price is evidently paid for subsumption repair due to the more aggressive repair process, compared to the one targeting equivalence violations.

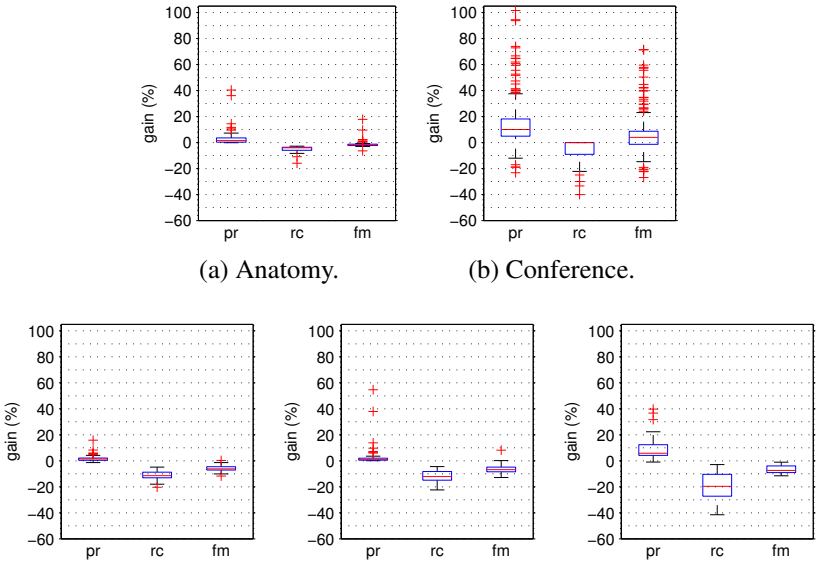
The correlation between the repair size and its impact is evident, but the two measures are not proportional. Consider the *library* track, the case with both the highest loss and the highest repair size, the removal of an average of 40% of the original mappings, leads to a $\sim 20\%$ recall loss (as median value), and $\sim 9\%$ for f-measure.

[9] Note, however, that the size of the consistency repair is usually extremely limited for the employed dataset.

In addition, we always need to take into consideration that the measures are computed against unrepaired reference alignments, that is the worst-case scenario for our repair algorithm. A more appropriate comparison would be against reference alignments where all the true positive conservativity violations are repaired. This additional set of expertiments were not performed in the context of the present thesis, due to the lack of domain experts able to cover all the different involved topics, and due to the difficulty in finding volunteer experts willing to devote a considerable time for the inspection of the alignments and ontologies, some of them having considerable size (in particular for *largebio* and *library* tracks). We consider these extended experiments an interesting research line that we aim at pursue as future work.

As a side note we remark that LogMapC, implementing the conservativity repair facility described in the present chapter, obtained the best average precision for the *largebio* track, in the 2014 edition of the *OAEI*.[10]

Results aggregated by matcher are available in [Sol15], Section 2.2.



(a) Anatomy. (b) Conference.

(c) LargeBio (Big). (d) LargeBio (Small). (e) Library.

Figure 6.11: Subsumption repair effect on precision, recall and f-measure.

[10] As reported in the results for the 2014 edition of the *largebio* track: `http://www.cs.ox.ac.uk/isg/projects/SEALS/oaei/2014/results2014_top.html`

6.5.5 Support for Manual Alignment Debugging

In this section, we analyze the GUI meant to support manual alignments debugging by means of a step-by-step example.

The first step consists in loading an alignment and the associated pair of input ontologies. At this point, the tool computes (and list) the identified subsumption violations. A parameter allows us to control the violation kind to show, and it is also possible to restrict the visualization to direct violations only.

Then, the user can visualize any of the detected violations, listed in the rightmost part of the GUI (see Figure 6.12). For any violation, the user can ask to the program whether or not the violation is direct (see Figure 6.13).

The visualization consists of a graph-based reprentation of the involved concepts, as well as of their contextual information (*i.e.*, one level of sub/super concepts relation, from the classification DAG). This representation is shown in the central panel of the GUI.

After visualizing all the violations of interest, the user is now ready to select a subset of the detected violations that are recognized as true positives, and run the repair algorithm. The user can also quickly run the repair against the full set of detected violations. For any selected violation, the tool adds a disjointness clause, if the necessary safety checks are satisfied (see Definition 6.2), otherwise its addition is skipped. Despite this, the violation can still be solved as a side-effect of the repair process for other selected violations.

After computing the repair, the solved violations are highlighted with the green color (see Figure 6.14).

The repair process can be iteratively applied on the result of a previous repair step, until the user considers all the violations as solved, or none of the remaining violations can be repaired by the algorithm.

Finally, the tool allows the user to save the alignment in the *Alignment API* format (RDF-based, supported also by the *OAEI*), that can be loaded and processed by any *Alignment API* compliant program.

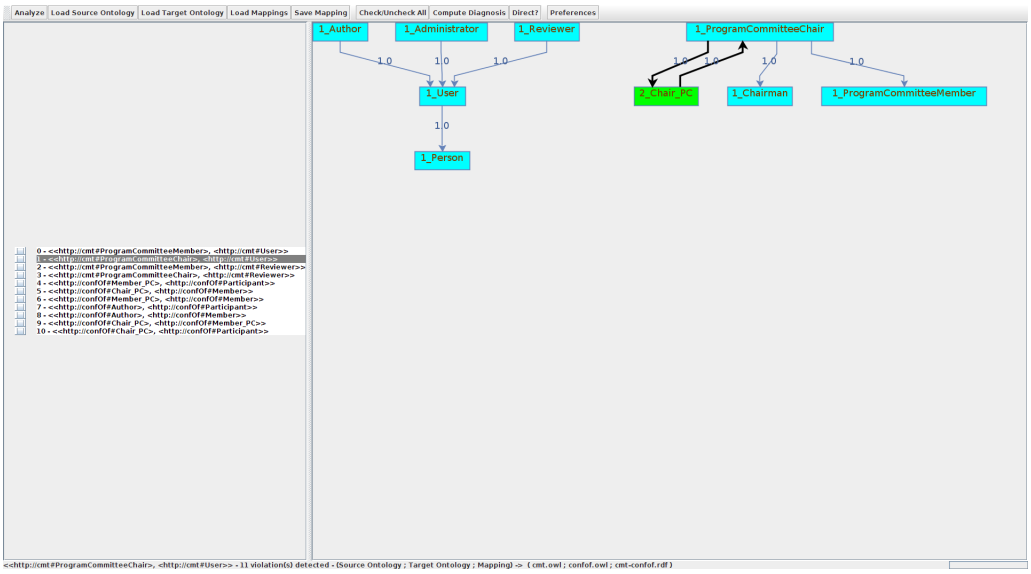


Figure 6.12: Visualization of a conservativity violation.

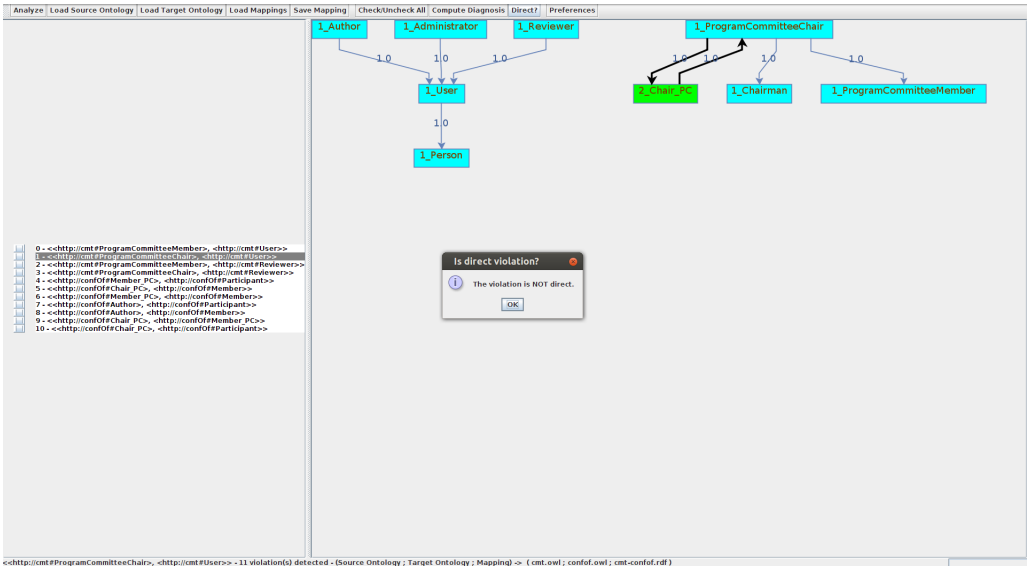


Figure 6.13: Direct violation test for a conservativity violation.

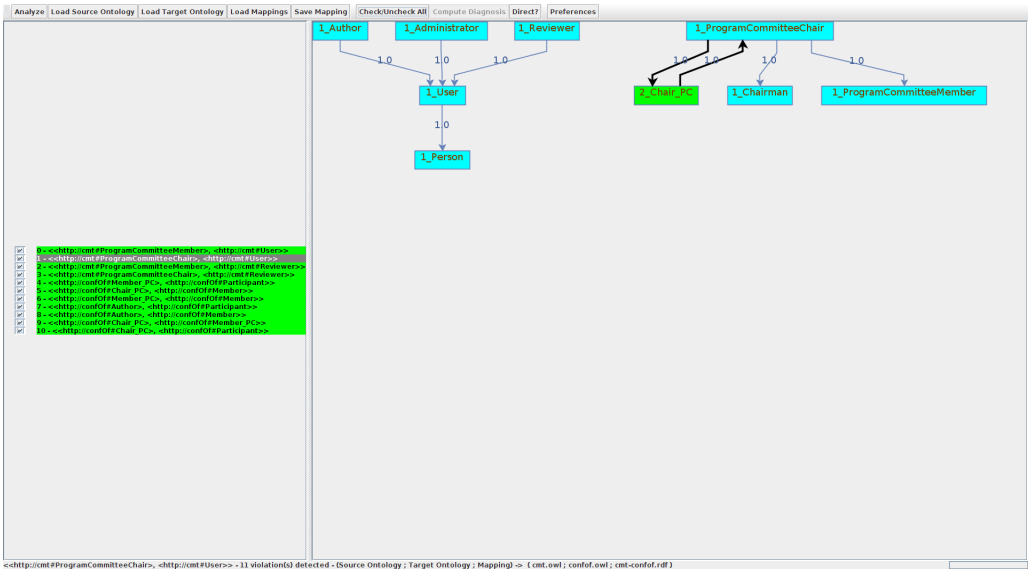


Figure 6.14: Repair status of the detected conservativity violation.

Chapter 7

A Combined Approach for Conservativity Principle Violations

7.1 Introduction

As already discussed, subsumption and equivalence conservativity violations present different intrinsic characteristics that are exploited by debugging algorithms for implementing efficient repair techniques. In order to be able to exploit these different characteristics, each repair algorithm is specifically tailored for a single violation kind.

For this reason, the need of a method able to cover all kinds of conservativity violations emerges, which is still not completely fulfilled by what proposed so far in the thesis.

With the aim of fully addressing the issue, the present chapter proposes a multi-strategy approach, appropriately combining the algorithms presented in Chapters 5 and 6.

In the remainder of the chapter we first introduce the multi-strategy approach, along with the detailed algorithm implementing it (Section 7.2). Then, the feasability of its application in practice is experimentally evaluated using the *OAEI* 2012–2014 dataset, and the results are shown in Section 7.3. Additional experimental results are available in [Sol15]. Given that the combined algorithm is presented in two variants, depending on the application order of the supported strategies, the experimental evaluation also aims at inferring the guidelines for choosing between the two proposed variants.

7.2 Combined Algorithm

As already discussed, the challenging number of conservativity violations requires to exploit the intrinsic characteristics of subsumption and equivalence violations, that result in the development of different approaches for their repair.

In Algorithm 23 we provide the repair algorithm combining the approaches targeting subsumption and equivalence violations. The algorithm takes as input the input ontologies, the alignment between them, and the Boolean parameter $sccFirst$, which controls the order in which the two (sub)repair algorithms are applied. As already discussed in Section 6.4, we again consider a preprocessing step which extracts the modules of the input ontologies w.r.t. the input alignment, and the full consistency repair of the input alignment itself. For this reason, when we refer to the input ontologies and alignment we implicitly refer to their modules and to its repaired version w.r.t. consistency, respectively.

When $sccFirst$ is true (line 3), we apply Algorithm 13 first, which addresses equivalence violations, and Algorithm 16 then, which is tailored for subsumption violations (line 7). Otherwise the repair order is reversed.

Given the empirical comparison of the efficiency and effectiveness of the different alternatives, we employ here the best performing ones. Concerning equivalence violations we therefore employ the *ASP* program for nonconservative diagnoses computation. Instead, for subsumption violations, we use the repair algorithm variant which optimizes the disjointness addition phase.

Finally, the repaired alignment and the approximate repair are returned.

Despite the differences between subsumption and equivalence violations, a mutual influence between different violation kinds exists. An example is given in Table 6.3 of Section 6.3.1, where (basic) subsumption violations are caused by the existence of equivalence violations in the same aligned ontology. Nonetheless, subsumption violations can be responsible for the existence of equivalence violations. Finally, also conservativity and consistency violations could influence each others, but this is out of the scope of the thesis, and we therefore always assume coherent alignments in order to separate the two problems.

It is evident, from the aforementioned considerations that the application of a repair step targeting a particular violation kind can solve, as a side-effect, also violations of other kinds.

For this reason, we expect that the application order of the different repair algorithm influences the repair performance in terms of required time and number of solved violations.

Algorithm 23 ComboRepair algorithm for detecting and solving conservativity principle violations

Input: $\mathcal{O}_1, \mathcal{O}_2$: input ontologies; $\mathcal{M}$: input mappings; $sccFirst$: Boolean flag;
Output: $\mathcal{M}'$: output mappings; $\mathcal{R}^{\approx}$: approximate repair

1: $\mathcal{R}^{\approx} \leftarrow \emptyset$
2: $\mathcal{M}' \leftarrow \mathcal{M}$
3: **if** $sccFirst$ **then**
4: $\quad \mathcal{R}^{\approx} \leftarrow \mathsf{CycleBreaker}(\mathcal{O}_1, \mathcal{O}_2, \mathcal{M}')$ ▷ Algorithm 13
5: $\quad \mathcal{M}' \leftarrow \mathcal{M}' \setminus \mathcal{R}^{\approx}$
6: **end if**
7: $\langle \mathcal{M}', \mathcal{R}^{\approx\prime} \rangle \leftarrow \mathsf{SubRepair}(\mathcal{O}_1, \mathcal{O}_2, \mathcal{M}', \mathsf{true}, \mathsf{false}, \mathsf{true})$ ▷ Algorithm 16
8: $\mathcal{R}^{\approx} \leftarrow \mathcal{R}^{\approx} \cup \mathcal{R}^{\approx\prime}$
9: **if not** $sccFirst$ **then**
10: $\quad \mathcal{R}^{\approx} \leftarrow \mathcal{R}^{\approx} \cup \mathsf{CycleBreaker}(\mathcal{O}_1, \mathcal{O}_2, \mathcal{M}')$ ▷ Algorithm 13
11: $\quad \mathcal{M}' \leftarrow \mathcal{M}' \setminus \mathcal{R}^{\approx}$
12: **end if**
13: **return** $\langle \mathcal{M}', \mathcal{R}^{\approx} \rangle$

7.3 Experimental Evaluation

In this section, we evaluate the feasibility of using our combined method to correct conservativity principle violations in practice. Section 7.3.1 analyzes the behaviour of our repair algorithm on the dataset composed by alignments computed by matching systems participating at the *OAEI* 2012–2014. Section 7.3.2 analyzes and compares the impact of conservativity repair strategies on the alignment quality in terms of precision, recall, and f-measure.

The test environment consists of a desktop computer equipped with 32*GB DDR*3 *RAM* at 1333*MHz*, and an *AMD Fusion FX 4350* (quad-core, each running at 4.2*GHz*) as *CPU*. The dataset is stored on a 128*GB SSD*, where the operating system (*Ubuntu* 12.04, 64-bit version) is also installed. As already discussed in Section 6.5.3, 26GB of RAM are allocated to the JVM in order to minimize the influence of the garbage collector on the recorded temporal measurements.

7.3.1 Analysis of Ontology Matchers Alignments

Table 7.1 (resp. Table 7.4) shows the size of the evaluated ontologies and mappings (**I**, **II** and **III**), and the number of original violations (**IV**, **V** and **VI**), with results grouped by matcher (resp. track). Last column shows the number of alignments contributing to each row (as already discussed in Section 6.5.3).

Tables 7.2 and 7.3, instead, relate to solution size and show the number of disjointness clauses added by the subsumption repair algorithm (resp. **VII** and **XIV**), the global size of the computed repair (resp. **IX** and **XVI**). Additionally, the tables show the time spent by the subsumption repair algorithm for detecting and adding the necessary disjointess clauses to the Horn projections of the input ontologies (resp. **VIII** and **XV**). The tables also report the global repair time (resp. **X** and **XVII**), obtained by summing the repair time of the two sub-methods. Finally, the number of

Algorithm 24 Conducted evaluation for combined algorithm

Input: $\mathcal{O}_1, \mathcal{O}_2$: input ontologies $\mathcal{M}$: alignment between $\mathcal{O}_1$ and $\mathcal{O}_2$

$\mathcal{O}^{\mathcal{M}} \leftarrow \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$ ▷ We assume the preprocessing step for module extraction and full consistency repair
Store size of $\mathsf{Sig}(\mathcal{O}_1)$ (**I**), $\mathsf{Sig}(\mathcal{O}_2)$ (**II**) and $\mathcal{M}$ (**III**)
Update $\mathcal{M}$ after full consistency repair
Compute number of conservativity principle violations:
$\mathsf{basicViol} \leftarrow |\mathsf{basicViol}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{basicViol}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**IV**) ▷ basic violations, Definition 4.26, Section 4.3.6
$\mathsf{diff}^{\approx} \leftarrow |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_1)}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_2)}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**V**) ▷ general notion, Definition 4.23, Section 4.3.4
$\mathsf{eqViol} \leftarrow |\mathsf{eqViol}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{eqViol}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**VI**) ▷ equivalence violations, Definition 4.26, Section 4.3.6
Compute repair $\mathcal{R}^{\approx}{}_1$ using Algorithm 23 with $sccFirst$ set to true (equivalence repair first, subsumption repair then)
Store number of added disjointness $disj$ (**VII**) and time to compute disjointness rules t_d (**VIII**),
Store size of (global) repair $|\mathcal{R}^{\approx}{}_1|$ (**IX**) and time to compute the (global) mapping repair t_r (**X**)
Compute repair $\mathcal{R}^{\approx}{}_2$ using Algorithm 23 with $sccFirst$ set to false (subsumption repair first, equivalence repair then)
Store number of added disjointness $disj$ (**XIV**) and time to compute disjointness rules t_d (**XV**),
Store size of (global) repair $|\mathcal{R}^{\approx}{}_2|$ (**XVI**) and time to compute the (global) mapping repair t_r (**XVII**)
$\mathcal{O}^{\mathcal{M}}_{\mathcal{O}_1,\mathcal{O}_2} \leftarrow \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M} \setminus \mathcal{R}^{\approx}{}_1$
Compute number of remaining conservativity principle violations:
$\mathsf{basicViol} \leftarrow |\mathsf{basicViol}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{basicViol}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**XI**)
$\mathsf{diff}^{\approx} \leftarrow |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_1)}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_2)}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**XII**)
$\mathsf{eqViol} \leftarrow |\mathsf{eqViol}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{eqViol}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**XIII**)
$\mathcal{O}^{\mathcal{M}} \leftarrow \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M} \setminus \mathcal{R}^{\approx}{}_2$
Compute number of remaining conservativity principle violations:
$\mathsf{basicViol} \leftarrow |\mathsf{basicViol}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{basicViol}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**XVIII**)
$\mathsf{diff}^{\approx} \leftarrow |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_1)}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{diff}^{\approx}_{\mathsf{Sig}(\mathcal{O}_2)}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**XIX**)
$\mathsf{eqViol} \leftarrow |\mathsf{eqViol}(\mathcal{O}_1, \mathcal{O}^{\mathcal{M}})| + |\mathsf{eqViol}(\mathcal{O}_2, \mathcal{O}^{\mathcal{M}})|$ (**XX**)

unsolved violations for the different kinds are reported (resp. **XI** and **XVIII**, **XII** and **XIX**, **XII** and **XX**).

Roman numbers refer to the stored measurements during the conducted evaluation, that is summarized in Algorithm 24.

Table 7.2 reports the results obtained with $sccFirst$ set to true (therefore applying the equivalence violation repair algorithm first, and the subsumption repair then), while Table 7.3 is associated with the execution of the algorithm with $sccFirst$ flag set to false.

Tables 7.5 and 7.6 are totally analogous to Tables 7.2 and 7.3, respectively, but they aggregate results by track (while Tables 7.2 and 7.3 aggregate the results by matcher).

Different groupings for the same experimental values are provided in [Sol15], Section 3.1.

The experimental results can be summarized as follows.

(i) From Tables 7.2, 7.3, 7.5 and 7.6 we can see that both variants of the combined repair algorithm are effective, because the quantity of unsolved violations is extremely reduced (w.r.t. any violation notion). For the *anatomy* and *library* tracks, all the basic and equivalence violations are solved, and restricting to equivalence violations only, the same also holds for the *conference* track. Considering the approximation of the deductive difference, the number of unsolved violations can be considered practically irrelevant.

(ii) By comparing the unsolved violations with that of the approaches targeting a single viola-

	Problem Size			Original Violations			Alignments
	I	II	III	IV	V	VI	
	$\lvert\mathsf{Sig}(\mathcal{O}_1)\rvert$	$\lvert\mathsf{Sig}(\mathcal{O}_2)\rvert$	$\lvert\mathcal{M}\rvert$	basicViol	diff$^{\approx}$	eqViol	$\#\mathcal{M}$
aml	12,224(29,889)	10,806(27,190)	3,462(7,054)	327,899(2,030,773)	331,163(2,042,411)	23,359(178,169)	59
amlbk	12,527(30,946)	10,936(28,172)	3,235(7,146)	46,468(120,759)	47,797(124,906)	56(214)	28
amlbkr	57,614(45,446)	50,061(44,114)	13,944(8,532)	201,763(220,499)	207,974(229,734)	290(467)	6
amlbku	57,614(45,446)	50,061(44,114)	19,944(13,448)	394,605(308,040)	403,076(314,020)	2,146(2,230)	6
amlbkur	57,614(45,446)	50,061(44,114)	18,907(12,874)	486,519(455,149)	499,143(470,545)	2,383(2,470)	6
amlr	57,614(45,446)	50,061(44,114)	13,359(8,405)	185,132(220,000)	190,467(228,646)	240(395)	6
aot	391(933)	560(1,501)	692(1,872)	2,512(7,807)	2,533(7,867)	113(439)	22
aotl	382(912)	539(1,469)	110(327)	430(1,486)	432(1,490)	7.87(19)	23
aroma	5,540(14,285)	7,336(24,043)	2,042(5,072)	41,521(114,796)	42,356(117,442)	38(99)	27
ase	107(27)	123(44)	47(27)	64(136)	66(137)	9.75(18)	20
autom	274(770)	413(1,372)	179(768)	282(1,321)	283(1,327)	5.68(27)	22
cidercl	237(576)	270(697)	208(737)	2,274(10,284)	2,288(10,346)	4.86(21)	21
codi	506(1,433)	618(1,817)	419(1,445)	12,514(59,481)	12,750(60,612)	8.74(41)	23
cromatcher	107(36)	121(46)	81(22)	26(38)	27(38)	0.27(1.03)	15
gomma	17,023(33,634)	15,225(31,294)	4,576(7,354)	547,306(3,129,072)	550,954(3,144,329)	55,801(338,465)	37
gommabk	38,540(42,323)	35,059(41,568)	16,185(11,274)	181,706(191,482)	191,265(202,308)	1,306(1,343)	7
gommasbk	57,614(45,446)	50,061(44,114)	18,493(10,722)	237,553(195,121)	248,350(204,851)	1,517(1,358)	6
hertuda	1,021(2,411)	1,368(3,245)	1,205(3,056)	1,070,035(5,279,337)	1,074,438(5,300,895)	154,430(762,715)	50
hotmatch	1,021(2,411)	1,368(3,245)	661(1,562)	4,356(18,533)	4,442(18,921)	2.44(11)	50
iama	12,321(30,409)	10,848(27,669)	1,722(4,455)	11,534(34,419)	11,796(35,326)	57(171)	29
logmap	11,362(28,458)	10,158(26,202)	3,332(6,999)	65,509(179,944)	67,388(185,803)	258(609)	88
logmap2noe	44,505(43,018)	40,351(42,874)	14,837(9,642)	262,095(311,406)	269,874(322,187)	1,009(900)	6
logmapbio	49,776(46,381)	43,382(43,977)	13,713(9,794)	358,123(410,317)	367,913(425,000)	1,351(1,256)	7
logmapbk	57,614(45,446)	50,061(44,114)	14,654(9,536)	313,796(342,009)	321,864(353,585)	1,073(995)	6
logmapc	12,321(30,409)	10,848(27,669)	2,126(4,523)	403(1,834)	478(2,052)	17(53)	29
logmaplt	11,417(28,618)	10,178(26,353)	2,662(6,201)	338,820(1,671,623)	342,082(1,684,896)	16,551(87,248)	87
maasmatch	607(1,629)	831(2,281)	917(2,622)	157,203(1,223,543)	158,621(1,234,257)	7,136(59,956)	71
mapsss	2,258(8,392)	1,832(5,379)	867(3,004)	12,953(65,926)	13,300(67,809)	13(49)	49
medley	110(31)	121(44)	98(43)	92(185)	93(185)	3.62(6.99)	21
odgoms	870(2,306)	1,182(3,133)	777(2,295)	12,249(63,650)	12,413(64,510)	59(362)	48
omreasoner	5,857(18,402)	4,539(14,089)	927(2,929)	4,876(15,362)	4,965(15,714)	47(218)	25
ontok2	110(31)	121(44)	18(6.73)	0.29(0.9)	0.29(0.9)	0	21
optima	506(1,433)	618(1,817)	179(500)	4,259(18,636)	4,340(18,994)	25(91)	23
rimom	110(31)	121(44)	114(48)	134(105)	139(108)	32(31)	21
rsdlwb	640(1,547)	866(2,150)	168(473)	76(213)	77(215)	0.08(0.41)	24
servomap	10,965(27,944)	9,844(25,912)	3,215(6,862)	148,063(565,523)	150,113(571,146)	1,371(9,139)	58
servomapl	9,609(25,712)	8,839(24,477)	3,233(6,898)	122,405(358,776)	124,813(364,640)	500(1,952)	29
sphere	57,614(45,446)	50,061(44,114)	9,331(7,650)	102,947(128,423)	105,714(132,577)	255(281)	6
stringsauto	640(1,547)	866(2,150)	358(968)	752(2,223)	759(2,241)	2.46(7.68)	24
synthesis	110(31)	121(44)	17(6.79)	0.33(0.91)	0.33(0.91)	0	21
toast	2,747(0)	3,306(0)	2,678(0)	2,597(0)	2,612(0)	4(0)	1
wesee	371(1,094)	446(1,380)	245(974)	2,977(18,757)	3,043(19,186)	1.07(5.76)	45
wmatch	308(757)	405(1,149)	258(1,037)	384(2,324)	390(2,356)	16(85)	45
xmap	12,321(30,409)	10,848(27,669)	3,852(7,909)	902,978(4,521,677)	908,611(4,541,750)	114,704(617,312)	29
xmapgen	473(1,620)	630(2,221)	242(811)	1,328(4,899)	1,340(4,941)	61(256)	45
xmapsig	605(1,838)	799(2,475)	267(823)	3,001(16,040)	3,093(16,613)	45(192)	46
yam++	10,965(27,944)	9,844(25,912)	3,375(6,945)	133,905(476,507)	136,107(482,130)	784(4,516)	58

Table 7.1: Measures related to problem size for *OAEI* 2012-2014 dataset, grouped by matcher.

	Solution Size		Times		Remaining Violations		
	VII	IX	VIII	X	XI	XII	XIII
	#disj	$\lvert\mathcal{R}^{\approx}{}_1\rvert$	t_d(s)	t_r(s)	basicViol	diff$^{\approx}$	eqViol
aml	29,465(76,197)	892(1,933)	9.35(22)	30(83)	8.27(31)	21(55)	0.08(0.38)
amlbk	27,153(80,068)	669(1,667)	7.74(18)	32(90)	5.36(21)	19(52)	0.11(0.42)
amlbkr	121,438(163,351)	3,157(2,566)	31(21)	137(151)	32(41)	88(77)	0.5(0.84)
amlbku	141,976(101,030)	6,007(4,497)	35(28)	167(159)	33(41)	214(180)	1.33(2.07)
amlbkur	169,378(160,769)	6,451(5,242)	42(37)	239(267)	122(196)	324(241)	1(1.55)
amlr	120,036(168,514)	2,910(2,411)	32(21)	132(144)	24(41)	71(63)	0.33(0.82)
aot	1,428(4,797)	163(501)	1.44(5.25)	0.22(0.67)	59(275)	60(278)	0.64(2.98)
aotl	257(1,014)	17(45)	1(3.35)	0.05(0.16)	0.52(2.5)	0.52(2.5)	0
aroma	25,408(86,319)	424(1,126)	5.98(15)	16(55)	4.11(20)	14(40)	0.07(0.38)
ase	24(38)	8.15(8.97)	0.11(0.31)	0.01(0.01)	0.35(0.99)	0.35(0.99)	0
autom	240(1,124)	22(101)	2.65(12)	0.07(0.3)	1.73(7.88)	1.73(7.88)	0
cidercl	1,114(4,970)	41(169)	0.22(0.52)	0.1(0.43)	0.52(1.66)	0.67(1.74)	0
codi	8,431(39,901)	94(406)	0.41(1.65)	1.85(8.76)	0.04(0.21)	0.35(1.11)	0
cromatcher	26(39)	8.39(6.6)	0.15(0.36)	0.01(0)	0.13(0.52)	0.13(0.52)	0
gomma	21,932(47,251)	970(1,782)	18(36)	30(65)	9.54(32)	25(47)	0.16(0.5)
gommabk	75,385(67,538)	3,825(2,938)	24(20)	80(92)	20(41)	131(130)	0
gommasbk	84,729(63,469)	4,400(2,738)	29(18)	107(103)	27(46)	145(108)	0.67(1.63)
hertuda	7,745(31,457)	624(1,795)	9.45(41)	9.74(25)	4.08(20)	4.4(20)	0
hotmatch	4,044(17,006)	93(276)	1.27(3.7)	1.03(3.88)	4.44(22)	4.76(22)	0
iama	6,808(23,682)	276(799)	5.11(12)	13(37)	1.38(6.86)	6.34(18)	0.03(0.19)
logmap	30,103(94,130)	904(2,047)	9.06(24)	33(103)	5.26(23)	17(49)	0.08(0.38)
logmap2noe	136,511(182,577)	3,959(3,157)	63(51)	161(200)	39(55)	87(61)	0.33(0.52)
logmapbio	124,634(172,578)	4,010(3,287)	34(31)	146(186)	17(41)	93(101)	0.29(0.76)
logmapbk	140,025(182,761)	4,033(3,195)	37(25)	174(207)	20(44)	82(82)	0.33(0.52)
logmapc	355(1,621)	32(113)	4.23(9.6)	8.14(24)	4.21(20)	21(55)	0.17(0.66)
logmaplt	17,555(49,580)	682(1,651)	9.63(23)	21(56)	9.86(34)	16(45)	0.02(0.15)
maasmatch	10,591(67,779)	194(716)	3.83(12)	5.18(30)	2.61(13)	3.86(17)	0
mapsss	10,329(58,451)	140(579)	4.97(24)	6.47(40)	5.84(35)	7.47(37)	0.08(0.45)
medley	56(89)	14(16)	0.34(0.74)	0.01(0.01)	0.67(2.11)	0.67(2.11)	0
odgoms	5,647(23,429)	159(580)	1.49(4.49)	1.62(6.7)	4.29(20)	4.75(21)	0
omreasoner	3,049(12,759)	140(457)	2.01(4.75)	3.2(11)	1.52(7.39)	2.4(8.44)	0
ontok2	0.29(0.9)	0.14(0.48)	0.04(0.05)	0	0.05(0.22)	0.05(0.22)	0
optima	1,748(7,712)	36(111)	0.21(0.52)	0.46(1.58)	0.04(0.21)	0.13(0.46)	0
rimom	60(65)	49(39)	0.85(2.98)	11(23)	0.05(0.22)	0.05(0.22)	0
rsdlwb	75(210)	11(33)	0.7(2.88)	0.08(0.22)	0.04(0.2)	0.12(0.45)	0
servomap	30,673(72,597)	804(1,857)	13(33)	40(121)	6.14(24)	18(42)	0.07(0.32)
servomapl	29,739(68,320)	724(1,587)	15(40)	33(98)	9(29)	24(54)	0.24(0.91)
sphere	50,384(71,062)	1,564(1,594)	29(20)	79(86)	18(41)	148(167)	0
stringsauto	731(2,160)	44(122)	0.88(3.6)	0.17(0.52)	0.17(0.56)	0.29(0.69)	0
synthesis	0.33(0.91)	0.24(0.62)	0.04(0.05)	0.01(0)	0.05(0.22)	0.05(0.22)	0
toast	2,556(0)	309(0)	0.62(0)	0.43(0)	0	0	0
wesee	2,737(17,885)	46(242)	0.18(0.7)	0.59(3.81)	0.04(0.21)	0.13(0.46)	0
wmatch	262(1,553)	41(199)	1.53(9.78)	1.27(8.07)	2.47(16)	2.58(16)	0
xmap	46,717(126,905)	998(2,216)	17(43)	45(121)	4.14(19)	17(38)	0.07(0.37)
xmapgen	603(2,572)	43(155)	0.67(2.86)	1.51(9.62)	3.4(23)	3.44(22)	0
xmapsig	1,148(5,411)	49(167)	0.69(2.77)	1.59(9.36)	3.33(22)	3.37(22)	0
yam++	28,923(66,203)	817(1,739)	12(30)	31(82)	6.14(24)	19(45)	0.03(0.18)

Table 7.2: Measures related to solution size for *OAEI* 2012-2014 dataset, grouped by matcher. The repair order is equivalence violations first, subsumption then.

	Solution Size		Times		Remaining Violations		
	XIV	XVI	XV	XVII	XVIII	XIX	XX
	#disj	$\lvert\mathcal{R}^{\approx}{}_{\mathbf{2}}\rvert$	$\mathbf{t}_d$(s)	$\mathbf{t}_r$(s)	basicViol	diff$^{\approx}$	eqViol
aml	327,899(2,030,773)	892(1,930)	11(26)	142(687)	6.83(26)	20(51)	0.12(0.59)
amlbk	46,468(120,759)	666(1,653)	9.4(24)	49(134)	5.18(21)	19(54)	0.11(0.42)
amlbkr	201,763(220,499)	3,184(2,583)	35(24)	183(188)	25(42)	76(76)	0.5(0.84)
amlbku	394,605(308,040)	6,166(4,580)	50(45)	375(396)	20(41)	186(151)	1.5(2.51)
amlbkur	486,519(455,149)	6,592(5,219)	61(79)	613(702)	101(192)	277(211)	1.67(3.2)
amlr	185,132(220,000)	2,941(2,435)	33(22)	173(185)	18(41)	57(54)	0.33(0.82)
aot	2,512(7,807)	168(524)	1.48(5.29)	0.36(1.18)	59(275)	59(278)	0.64(2.98)
aotl	430(1,486)	18(48)	1(3.35)	0.05(0.18)	0.52(2.5)	0.52(2.5)	0
aroma	41,521(114,796)	428(1,137)	8.51(26)	36(128)	4.11(20)	13(40)	0.11(0.58)
ase	64(136)	8.4(9.48)	0.11(0.31)	0.01(0.01)	0.15(0.49)	0.15(0.49)	0
autom	282(1,321)	22(104)	2.45(11)	0.06(0.27)	1.73(7.88)	1.73(7.88)	0
cidercl	2,274(10,284)	43(177)	0.22(0.54)	0.2(0.88)	0.52(1.66)	0.67(1.74)	0
codi	12,514(59,481)	95(410)	0.43(1.71)	2.39(11)	0.04(0.21)	0.22(0.6)	0
cromatcher	26(38)	8.41(6.59)	0.15(0.36)	0.01(0.01)	0.13(0.52)	0.13(0.52)	0
gomma	547,306(3,129,072)	982(1,785)	20(41)	221(1,092)	9.43(32)	27(52)	0.16(0.5)
gommabk	181,706(191,482)	3,851(2,925)	59(61)	165(211)	40(48)	152(136)	0
gommasbk	237,553(195,121)	4,428(2,690)	38(30)	215(226)	32(49)	154(148)	0
hertuda	1,070,035(5,279,337)	617(1,723)	11(49)	490(2,406)	4.08(20)	4.96(20)	0
hotmatch	4,356(18,533)	93(276)	1.34(3.96)	1.11(4.17)	4.44(22)	4.76(22)	0
iama	11,534(34,419)	267(771)	6.33(16)	17(52)	1.31(6.87)	7.31(22)	0.07(0.26)
logmap	65,509(179,944)	917(2,078)	11(28)	71(230)	5.09(23)	17(48)	0.15(0.58)
logmap2noe	262,095(311,406)	3,962(3,143)	77(67)	303(370)	37(56)	90(75)	0.5(0.84)
logmapbio	358,123(410,317)	4,069(3,330)	43(45)	397(537)	16(42)	77(83)	0.71(1.25)
logmapbk	313,796(342,009)	4,081(3,281)	43(37)	383(488)	18(45)	78(87)	0.5(0.84)
logmapc	403(1,834)	33(115)	4.88(12)	8.48(25)	4.21(20)	21(55)	0.17(0.66)
logmaplt	338,820(1,671,623)	691(1,665)	13(30)	136(585)	5.37(22)	13(34)	0.07(0.25)
maasmatch	157,203(1,223,543)	208(781)	4.2(14)	25(184)	2.69(13)	4.1(17)	0
mapsss	12,953(65,926)	142(583)	4.96(24)	6.77(40)	5.84(35)	7.35(36)	0.08(0.45)
medley	92(185)	14(14)	0.34(0.75)	0.01(0.01)	0.24(0.89)	0.24(0.89)	0
odgoms	12,249(63,650)	165(616)	1.54(4.56)	3.21(16)	4.29(20)	4.79(21)	0
omreasoner	4,876(15,362)	144(466)	2.51(5.91)	4.63(14)	1.52(7.39)	2.4(8.44)	0
ontok2	0.29(0.9)	0.14(0.48)	0.04(0.05)	0.01(0)	0.05(0.22)	0.05(0.22)	0
optima	4,259(18,636)	40(121)	0.26(0.76)	0.6(2.7)	0.04(0.21)	0.13(0.46)	0
rimom	134(105)	49(39)	0.84(2.92)	0.19(0.62)	0.05(0.22)	0.05(0.22)	0
rsdlwb	76(213)	11(33)	0.76(3.19)	0.08(0.23)	0.04(0.2)	0.12(0.45)	0
servomap	148,063(565,523)	879(2,035)	18(48)	125(364)	5.74(23)	15(34)	0.05(0.29)
servomapl	122,405(358,776)	724(1,568)	19(46)	83(238)	8.31(29)	20(48)	0.24(0.91)
sphere	102,947(128,423)	1,581(1,604)	32(23)	114(121)	18(41)	165(185)	0
stringsauto	752(2,223)	45(123)	0.92(3.72)	0.18(0.52)	0.17(0.56)	0.29(0.69)	0
synthesis	0.33(0.91)	0.24(0.62)	0.04(0.05)	0.01(0)	0.05(0.22)	0.05(0.22)	0
toast	2,597(0)	311(0)	0.63(0)	0.39(0)	0	0	0
wesee	2,977(18,757)	46(243)	0.19(0.76)	0.68(4.29)	0.04(0.21)	0.29(1.38)	0
wmatch	384(2,324)	43(211)	1.52(9.75)	0.07(0.34)	2.47(16)	2.58(16)	0
xmap	902,978(4,521,677)	979(2,155)	20(53)	470(2,216)	4.07(19)	15(33)	0.07(0.37)
xmapgen	1,328(4,899)	46(163)	0.7(3)	0.21(1.01)	3.4(23)	3.44(22)	0
xmapsig	3,001(16,040)	53(176)	0.73(2.87)	0.62(3.01)	3.33(22)	3.52(22)	0
yam++	133,905(476,507)	824(1,760)	14(35)	80(215)	5.72(24)	17(39)	0.07(0.37)

Table 7.3: Measures related to solution size for *OAEI* 2012-2014 dataset, grouped by matcher. The repair order is subsumption violations first, equivalence then.

	Problem Size			Original Violations			Alignments
	I	II	III	IV	V	VI	
	$\lvert\mathsf{Sig}(\mathcal{O}_1)\rvert$	$\lvert\mathsf{Sig}(\mathcal{O}_2)\rvert$	$\lvert\mathcal{M}\rvert$	basicViol	$\mathsf{diff}^{\approx}$	eqViol	$\#\mathcal{M}$
anatomy	2,747(0)	3,306(0)	2,607(816)	6,562(12,844)	6,619(12,911)	137(340)	47
conference	110(30)	121(43)	33(35)	17(86)	17(87)	1.27(7.74)	1,104
largebio_big	18,340(19,857)	13,450(6,995)	11,789(8,729)	154,566(237,174)	159,711(245,746)	716(1,100)	122
largebio_small	84,567(30,642)	79,217(33,050)	13,868(9,056)	270,491(268,151)	276,684(274,228)	822(1,070)	91
library	6,575(0)	8,376(0)	6,363(3,168)	5,121,627(8,375,902)	5,153,135(8,413,450)	516,289(1,125,536)	32

Table 7.4: Measures related to problem size for *OAEI* 2012-2014 dataset, grouped by track.

	Solution Size		Times		Remaining Violations		
	VII	IX	VIII	X	XI	XII	XIII
	#disj	$\lvert\mathcal{R}^{\approx}{}_1\rvert$	$\mathbf{t}_d$(s)	$\mathbf{t}_r$(s)	basicViol	$\mathsf{diff}^{\approx}$	eqViol
anatomy	4,011(7,888)	384(400)	0.76(0.4)	4.39(14)	0	1.6(0.85)	0
conference	11(42)	3.43(10)	0.16(1.41)	0.22(3.48)	0.16(1.24)	0.16(1.24)	0
largebio_big	110,323(153,602)	2,851(2,823)	24(17)	87(145)	52(130)	104(140)	0.43(1.45)
largebio_small	81,663(82,949)	3,134(2,638)	57(43)	157(137)	24(53)	98(113)	0.44(0.98)
library	90,154(53,457)	2,896(2,329)	39(62)	42(42)	0	3.81(4)	0

Table 7.5: Measures related to solution size for *OAEI* 2012-2014 dataset, grouped by track. The repair order is equivalence violations first, subsumption then.

	Solution Size		Times		Remaining Violations		
	XIV	XVI	XV	XVII	XVIII	XIX	XX
	#disj	$\lvert\mathcal{R}^{\approx}{}_2\rvert$	$\mathbf{t}_d$(s)	$\mathbf{t}_r$(s)	basicViol	$\mathsf{diff}^{\approx}$	eqViol
anatomy	6,562(12,844)	397(414)	0.77(0.4)	1.31(2.6)	0	1.51(0.91)	0
conference	17(86)	3.43(10)	0.17(1.45)	0.01(0.09)	0.15(1.33)	0.15(1.33)	0
largebio_big	154,566(237,174)	2,926(2,890)	25(18)	135(287)	53(129)	107(144)	0.51(1.48)
largebio_small	270,491(268,151)	3,159(2,643)	76(56)	344(353)	16(42)	78(96)	0.57(1.34)
library	5,121,627(8,375,902)	2,881(2,204)	48(75)	1,967(3,690)	0	15(10)	0

Table 7.6: Measures related to solution size for *OAEI* 2012-2014 dataset, grouped by track. The repair order is subsumption violations first, equivalence then.

tion kind (*i.e.*, Tables 5.6 and 5.7 for the equivalence repair algorithm, Tables 6.9 and 6.11 for the subsumption repair algorithm), we can see that the number of unsolved violations of both kinds is lowered when applying the combined repair approach.

(iii) The detection and disjointess addition time for the subsumption repair algorithm (**VIII** and **XV**) is comparable for both variants, and it is not therefore significantly influenced by the repair application order. In particular, the runtime for these tasks depends exclusively on the size of the input ontologies and alignment. The previous repair step affects only the alignment size, but the effects are limited due to the minimality of computed repairs for equivalence violations.

(iv) Total repair times t_r are extremely different and they are evidently influenced by the repair application order. The total repair time applying subsumption repair first, and equivalence repair then (**X**), is basically coinciding with that of applying subsumption repair in isolation (see Tables 6.9 and 6.11). The difference is lower than that required by the equivalence repair in isolation, because part of the problematic SCCs are solved as a side-effect of the subsumption repair. Instead, the total repair time applying equivalence repair first, and subsumption repair then (**XVII**), is sensibly lower than the repair time required by the other variant.

(v) The computed repairs (**IX** and **XVI**) are of comparable size, on average smaller when equivalence repair is applied first. The reason is that in most of the cases, *ASP*-based repair computation is exact, while the subsumption repair algorithm employs a heuristic computation. An exception is, however, represented by the *library* track, for which several hard cases represented by almost fully connected SCCs exist (due to the heavy frequency of multiple mappings for the same entity). In these cases, the heuristic-based subsumption repair performs better than the optimization features offered by the *ASP* engine.

This set of experiments confirms the increased effectiveness, with a small loss in efficiency, of the combined repair algorithm, w.r.t. the single repair methods. The two variants are comparable under any aspect, at the exception of the total repair time, on average lower for the variant applying equivalence repair first. The motivation lies in the fact that the subsumption algorithm, in presence of SCCs, generates a large number of almost equivalent repair plans, that pose a problem to the plan selection method. The solution based on *ASP*, instead, can focus on the problematic SCCs, and is it only indirectly affected by the subsumption violations (*i.e.*, they can influence the existence of some equivalence violations). However, in presence of a relevant number of multiple occurrence mappings, the variant applying subsumption repair first is, on average, more effective.

7.3.2 Repair Effects on Alignment Quality

In Figures 7.1–7.5 the average impact on precision, recall and f-measure, due to the application of the computed repairs is shown, analogously to what reported in Section 5.5.5 and Section 6.5.4 for equivalence and subsumption violations repair, respectively. For each pair of figures, on the left the result for subsumption repair first and equivalence repair then, is shown. The figure on the right, instead, reports the result for the opposite repair application order.

Again, the results are grouped by track (that is, *anatomy*, *conference*, *largebio* and *library*), and the two variants of *largebio* track (that is, *largebio-small* and *largebio-big*) are separately analyzed.

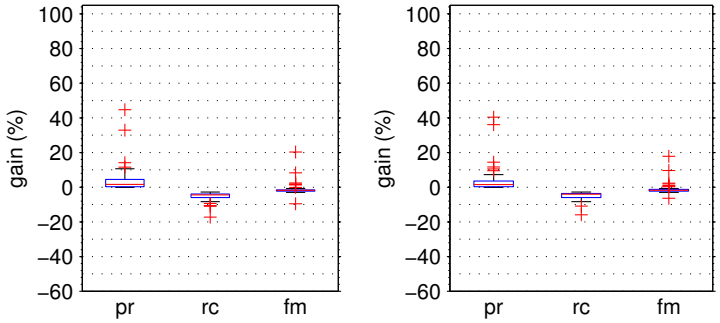
The impact of alignment repair is computed as the percentual of gain (resp. loss for negative values) for each measure computed for a repaired alignment, compared to the same measure computed for the original alignment. As in Section 6.5.4, the measures are computed w.r.t. the reference alignments provided by the *OAEI* organizers, and the original computed alignments are (fully) repaired w.r.t. consistency, as a preliminary step. Additionally, we again filter any alignment without conservativity violations, because the empty repair always implies a void gain.

Figures 7.1–7.5 confirm the general effects of a repair algorithm (*i.e.*, an increase of precision and a decrease of recall). The results achieved by the two versions of the combined repair algorithm are totally comparable. The gains for precision and recall result in an increase of the resulting f-measure for *conference* track, and a slight decrease for *anatomy* track. Instead, for both variants of *largebio* and *library* track, the loss is higher, and lies in the range $[0, 10]\%$.

From the results it is evident that the effect of the combined repair algorithm is in line with that of the repair algorithm addressing exclusively subsumption violations, in terms of performance w.r.t. a reference alignment. The correlation between the repair size and its impact is again evident.

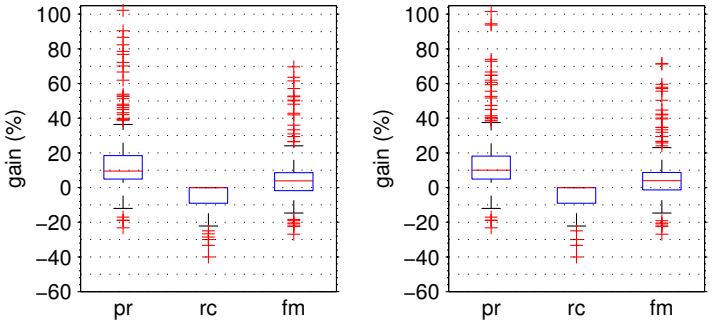
We also remind that the measures are computed against unrepaired reference alignments, while a more appropriate comparison would be against reference alignments where all the true positive conservativity violations are repaired. For instance, having the highest gains in f-measure for the *anatomy* and *conference* tracks, seems to confirm our hypothesis because, in these cases, the reduced size of the input ontologies and reference alignments allows for a more effective revision, which also limits the conservativity violations (see Table 6.5, where only 5 out 21 reference alignments for the *conference* track present more than 5 conservativity violations, and the reduced number of violations for the *anatomy* track), that are more frequent in the other tracks (considering of course also the difference in the size of the alignments). Finally, it is not surprising that the worst result in terms of the effect on the f-measure is obtained for the *library* track, for which the extremely high number of conservativity violations (see Table 6.5) highligths the need for a more accurate revision of the reference alignment.

Results aggregated by matcher are available in [Sol15], Section 3.2.



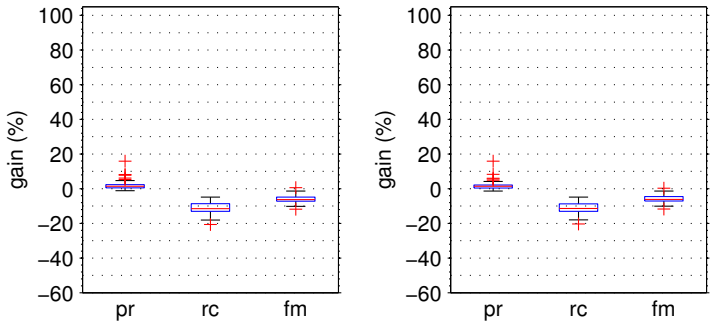
(a) Subsumption→Equivalence. (b) Equivalence→Subsumption.

Figure 7.1: Combined repair effects for *anatomy* track.



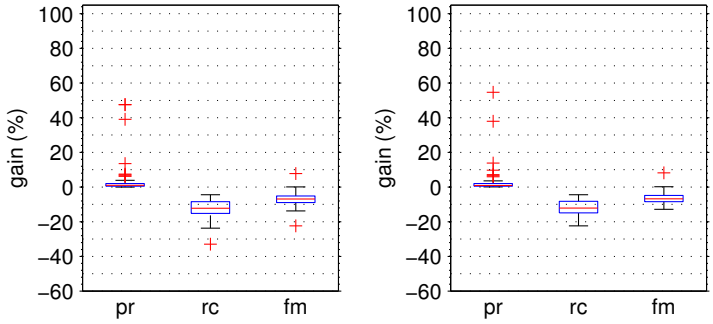
(a) Subsumption→Equivalence. (b) Equivalence→Subsumption.

Figure 7.2: Combined repair effects for *conference* track.



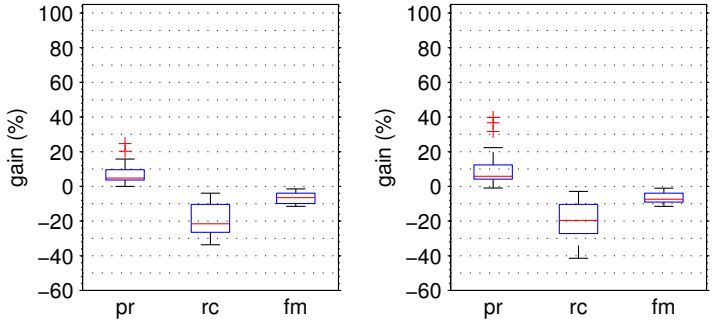
(a) Subsumption→Equivalence. (b) Equivalence→Subsumption.

Figure 7.3: Combined repair effects for *largebio-big* track.



(a) Subsumption→Equivalence. (b) Equivalence→Subsumption.

Figure 7.4: Combined repair effects for *largebio-small* track.



(a) Subsumption→Equivalence. (b) Equivalence→Subsumption.

Figure 7.5: Combined repair effects for *library* track.

Part II - Conclusions and Future Work

The second part of the thesis focused on change management for ontologies and related metadata, with a particular attention on semantic validation and debugging of ontology-to-ontology alignments. As already discussed, the presented techniques are far more general and despite being fundamental for implementing sound adaptations mechanisms (as shown in Chapter 4, Section 4.7), they are not constrained to be applied exclusively in the evolutionary context.

For the proposed approaches, different alternative techniques and strategies have been devised, with a comprehensive experimental comparison of their effectiveness and efficiency. We achieved this by means of the extensive dataset composed by alignments of the *OAEI* 2012–2014, comprising different ontology and alignment sizes, different ontology fragment expressiveness, and specific characteristics affecting the detection and repair algorithms.

The experiments proved the effectiveness and efficiency of the proposed techniques, and confirmed the different hypotheses underlying their design and optimizations.

The analysis of the reference and computed alignments in the aforementioned dataset highlighted the strong presence of conservativity violations. A study conducted by domain experts on the reference alignment of the anatomy track of the *OAEI* [BH12], analysing only trivial equivalence violations stemming from multiple mappings, confirmed as true positives half of the violations. It is reasonable that this rate increases in case of violations caused by complex interactions between mappings and local axioms, that are hard to be manually detected, and for this reason difficult to be foreseen by the alignment curators. Even considering as true positives approximately the 50% of the violations detected during our evaluation, the number would be extremely relevant.

We would also like to stress that the runtime of the proposed algorithms, as well as the minimality of the computed repairs, can only benefit from selecting and repairing only a subset of the detected violations, corresponding to the set of true positives.

In order to support the domain experts in the detection of conservativity violations, and the discrimination between true and false positives, we have developed prototypical GUIs, tailored for a selective and multi-step repair process, facilitated by the visualisation of contextual information for the concepts involved in the violations.

We have also identified various motivating scenarios in which the alignment is the only possible repair target, and we have provided a reference scenario, based on query answering tasks, involving multiple ontologies connected through ontology alignments, where the negative effects of conservativity violations, and the drawbacks of poor repair strategies, have been shown.

The contributions presented in the second part of the thesis can be further extended in the following directions.

The notion of direct violation has been used to summarize relevant information, thus allowing

a more effective user interaction. However, direct violations could be used for reducing the memory occupation of our approach, and to possibly speed-up the repair method based on the D&G algorithm, by enriching the projections of the input ontologies only with direct violations, instead of adding direct and derived ones.

Another interesting line of research would consist in the involvement of domain experts for a comprehensive evaluation of the rate of true/false positives in the analysed dataset, in order to precisely estimate it.

Additionally, we plan to apply supervised learning and linguistic-based techniques for the automatic classification of true/false positives, feature that would complement the already supported user interaction.

As discussed, the repair technique presented in Chapter 6 has been integrated into LogMap matcher, as a single-step at the end of the matching process. Our aim is to integrate the repair features as a multi-step process, similarly to what LogMap already implements for consistency repair, and explore the combined behaviour of the matching and repair techniques w.r.t. the reach of a *consesus* (*i.e.*, the maximal violations-free alignment, where any addition would cause a violation).

We already detailed in Section 4.7.3 the drawbacks of applying evolutionary approaches to ontology alignments, without integrating reasoning services. As already sketched, we consider the state-of-the-art proposals for alignment adaptation to be compatible with the repair techniques for consistency and conservativity violations, and we aim at proposing improved adaptation algorithms equipped with repair features.

Furthermore, we plan to explore the integration of repair techniques in all the other target scenarios identified in Section 4.7.

A further cross-fertilization between ontology alignment debugging and evolution fields, current detection and repair algorithms could be enriched to support the incremental update of both input ontologies and alignments. For instance, existing proposals for efficient maintenance of vertex reachability structures in digraphs [RZ04, RZ08] and for online update of SCCs [HKM+12] could be a promising starting point for such extension.

Finally, we also aim at supporting more expressive logical fragments while maintaining the nice scalability properties of the current algorithms.

Chapter 8

Conclusions

In the present thesis we addressed the problem of change management on relevant building blocks at the basis of the World Wide Web. Given the extremely wide range of data models composing the backbone of the Web, we decided to focus on subfields of the semi-structured and logic-based data management.

Concerning the first part of the thesis, we addressed two relevant problems in XML co-evolution, that is, an efficient validation of XML document adaptations upon schema changes, and XML document differencing.

The first problem allowed us to show the tight relationship between a document collection and the associated schema, and how changes to the latter deeply influences the former.

With the second problem we investigated the problem of finding a minimal set of operations for transforming a source XML document into a target one. The proposed technique, however, can also be used as an efficient change detection tool for triggering the refresh of materialized XML data or metadata, such as indexes.

These two scenarios do not only share the semi-structured data model of XML, but more generally are archetype of co-evolutionary approaches in which it is sufficient to consider the syntactycal and hierarchical levels, in order to ensure the correctness of the proposed adaptation stategies.

In the second part of the thesis, instead, we first provided examples of the insufficiency of a purely structural change management in logic-based context, using different concrete application scenarios, including ontology-to-ontology alignment adaptation upon ontology changes. In this context, we highlighted the drawbacks of pure syntactycal change management, showing the need for a tight integration of reasoning and logical repair techniques, inside the adaptation process.

Then, given the lack in the literature of a comprehensive study of the conservativity principle problem for ontology alignments, we investigated different alternative techniques for addressing it, both from a theoretical and an experimental point of view.

After the proposal of different techniques tailored for partial aspects of the problem, we finally derived a multi-strategy approach to combine them, with a practical evaluation of its increased effectiveness, at the cost of a modest loss in efficiency.

We therefore consider to have proposed relevant techniques in the context of change management for XML and ontology alignments, and to have contributed with concrete examples and reference scenarios to the known difference between traditional and logic-based management of data changes.

Appendix A

Change Ratio Estimation (extended results)

In this appendix we provide the whole detailed results for the pq-gram based change ratio estimation experiment, already discussed in Section 3.7.2. For all considered PUL types, the average estimated value and its standard deviation are reported in Tables A.1, A.2, A.3, and A.4.

Operation type	**0.01**	**0.05**	**0.1**	**0.15**	**0.2**
`ren` to a name already used in the document	0.07995(5.85e-03)	0.32298(1.36e-02)	0.52075(1.54e-02)	0.65080(1.28e-02)	0.74279(1.04e-02)
`ren` to a randomly generated name	0.08002(5.94e-03)	0.32379(1.38e-02)	0.52131(1.61e-02)	0.65244(1.37e-02)	0.74453(1.11e-02)
`ren` to a name of a node at the same nesting depth	0.07975(5.86e-03)	0.31904(1.17e-02)	0.51359(9.70e-03)	0.64291(7.18e-03)	0.73323(5.47e-03)
`ren` to a name of a node with the same parent name	0.07926(5.51e-03)	0.31959(1.04e-02)	0.51473(1.05e-02)	0.64307(7.26e-03)	0.73370(5.65e-03)
`ren` to a name of a node at the same nesting depth +- 1	0.07954(5.85e-03)	0.31956(9.98e-03)	0.51469(9.59e-03)	0.64364(6.74e-03)	0.73366(5.51e-03)
Move 1 tree (total weight 1–500)	0.00916(3.29e-03)	0.03824(1.23e-02)	0.07428(1.93e-02)	0.11179(2.78e-02)	0.14926(3.44e-02)
Move 2 trees (total weight 2–500)	0.00692(2.67e-03)	0.02466(9.98e-03)	0.04386(1.63e-02)	0.06433(2.37e-02)	0.08279(2.78e-02)
Move 3 trees (total weight 3–500)	0.00503(2.41e-03)	0.01606(7.02e-03)	0.02828(1.06e-02)	0.03953(1.43e-02)	0.05150(1.90e-02)
Move 4 trees (total weight 4–500)	0.00415(2.15e-03)	0.01104(5.35e-03)	0.01778(7.26e-03)	0.02592(8.38e-03)	0.03418(1.17e-02)
`repN` with a randomly generated tree of a similar weight	0.01551(1.18e-03)	0.07260(4.67e-03)	0.13833(7.77e-03)	0.19788(1.04e-02)	0.25219(1.22e-02)
`repN` with 1 randomly generated tree weighting 1–10	0.01563(1.40e-03)	0.07460(5.50e-03)	0.14119(8.40e-03)	0.20223(1.12e-02)	0.25708(1.34e-02)
`repN` with 1 randomly generated tree weighting 10–25	0.01598(1.22e-03)	0.07471(5.06e-03)	0.14014(8.87e-03)	0.19811(1.15e-02)	0.25017(1.32e-02)
`repN` with 1 randomly generated tree weighting 25–50	0.01663(1.27e-03)	0.07654(5.21e-03)	0.14313(8.76e-03)	0.20137(1.15e-02)	0.25254(1.38e-02)
`repN` with 1 randomly generated tree weighting 50–500	0.01664(1.21e-03)	0.07711(5.18e-03)	0.14327(8.92e-03)	0.20066(1.11e-02)	0.25091(1.34e-02)
`repN` with 2 randomly generated trees (total weight 50–100)	0.01666(1.14e-03)	0.07693(5.16e-03)	0.14317(9.04e-03)	0.20091(1.13e-02)	0.25155(1.34e-02)
`repN` with 3 randomly generated trees (total weight 75–150)	0.01673(1.09e-03)	0.07708(5.12e-03)	0.14347(8.82e-03)	0.20086(1.14e-02)	0.25179(1.32e-02)
`repN` with a tree similar to one at the same nesting depth	0.01227(1.00e-03)	0.05696(3.52e-03)	0.10741(6.74e-03)	0.15434(1.04e-02)	0.19753(1.28e-02)
`repN` with a tree similar to one of its siblings	0.01183(1.29e-03)	0.05449(5.00e-03)	0.10310(1.00e-02)	0.14977(1.27e-02)	0.19097(1.52e-02)
`repN` with 1 real tree	0.00985(7.98e-04)	0.04600(2.21e-03)	0.08709(4.20e-03)	0.12468(6.46e-03)	0.16000(7.88e-03)
`repN` with 2 real trees	0.00988(5.15e-04)	0.04637(1.85e-03)	0.08700(3.78e-03)	0.12541(4.66e-03)	0.15911(6.12e-03)
`repN` with 3 real trees	0.00994(4.28e-04)	0.04653(1.84e-03)	0.08780(3.39e-03)	0.12595(4.17e-03)	0.16071(5.06e-03)
`ins`$^{\rightarrow}$ 1 randomly generated tree (weight similar to a sibling)	0.01855(1.36e-03)	0.08565(6.43e-03)	0.15892(1.10e-02)	0.22223(1.51e-02)	0.27831(1.74e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 1–10	0.01867(1.39e-03)	0.08759(6.00e-03)	0.16210(9.99e-03)	0.22673(1.31e-02)	0.28315(1.52e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 10–25	0.01701(1.32e-03)	0.07970(5.46e-03)	0.14806(9.25e-03)	0.20734(1.19e-02)	0.25940(1.39e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 25–50	0.01702(1.29e-03)	0.07941(5.25e-03)	0.14724(9.10e-03)	0.20615(1.21e-02)	0.25736(1.38e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 50–500	0.01693(1.19e-03)	0.07768(5.44e-03)	0.14381(8.77e-03)	0.20140(1.15e-02)	0.25192(1.32e-02)
`ins`$^{\rightarrow}$ 2 randomly generated trees (total weight 50-100)	0.01683(1.26e-03)	0.07827(5.30e-03)	0.14523(8.91e-03)	0.20324(1.16e-02)	0.25385(1.35e-02)
`ins`$^{\rightarrow}$ 3 randomly generated trees (total weight 75-150)	0.01699(1.07e-03)	0.07798(5.22e-03)	0.14506(8.86e-03)	0.20250(1.13e-02)	0.25353(1.30e-02)
`ins`$^{\rightarrow}$ 1 tree similar to one at the same nesting depth	0.01437(1.33e-03)	0.06574(5.23e-03)	0.12262(9.04e-03)	0.17205(1.18e-02)	0.21649(1.42e-02)
`ins`$^{\rightarrow}$ 1 tree similar to one of its siblings	0.01392(1.57e-03)	0.06414(5.94e-03)	0.11942(1.06e-02)	0.16847(1.35e-02)	0.21258(1.80e-02)
`ins`$^{\rightarrow}$ 1 real tree	0.01079(7.05e-04)	0.04985(1.60e-03)	0.09382(2.55e-03)	0.13340(3.08e-03)	0.16942(3.33e-03)
`ins`$^{\rightarrow}$ 2 real trees	0.01036(5.25e-04)	0.04909(1.33e-03)	0.09305(1.96e-03)	0.13257(2.26e-03)	0.16931(2.67e-03)
`ins`$^{\rightarrow}$ 3 real trees	0.01032(5.06e-04)	0.04882(1.38e-03)	0.09276(1.70e-03)	0.13211(2.46e-03)	0.16874(2.37e-03)
`del` a node weighting 1-500	0.01440(1.65e-03)	0.06771(5.41e-03)	0.12969(8.92e-03)	0.18679(1.24e-02)	0.24169(1.48e-02)
`del` a node weighting 1-10	0.01603(1.77e-03)	0.07667(7.49e-03)	0.14556(1.39e-02)	0.20980(1.78e-02)	0.26751(2.14e-02)
`del` a node weighting 10-25	0.01155(1.48e-03)	0.05554(4.60e-03)	0.10808(7.97e-03)	0.15778(8.80e-03)	0.20647(1.00e-02)
`del` a node weighting 25-500	0.01021(1.01e-03)	0.05030(2.24e-03)	0.09951(2.43e-03)	0.14765(3.63e-03)	0.19675(6.05e-03)
Random	0.01859(3.47e-03)	0.07947(6.93e-03)	0.14489(1.17e-02)	0.20319(1.21e-02)	0.25369(1.51e-02)
`repV` with a value already used in the document	0.06350(4.46e-03)	0.26836(1.55e-02)	0.45024(2.02e-02)	0.58043(2.07e-02)	0.67752(1.81e-02)
`repV` with a randomly generated value	0.06373(4.35e-03)	0.27068(1.50e-02)	0.45484(1.95e-02)	0.58658(1.97e-02)	0.68531(1.74e-02)
`repV` with that of a node with the same name	0.06093(5.17e-03)	0.24376(2.51e-02)	0.39681(4.02e-02)	0.50116(4.85e-02)	0.57451(5.31e-02)

Table A.1: *pq*-gram based change ratio estimation (part 1).

Operation type	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**
`ren` to a name already used in the document	0.85890(6.39e-03)	0.92782(4.25e-03)	0.97396(4.44e-03)	0.99123(2.11e-03)	
`ren` to a randomly generated name	0.86100(6.80e-03)	0.93019(4.60e-03)	0.97652(5.06e-03)	0.99253(2.34e-03)	
`ren` to a name of a node at the same nesting depth	0.84835(5.87e-03)	0.91734(7.55e-03)	0.96378(6.52e-03)	0.98851(3.31e-03)	
`ren` to a name of a node with the same parent name	0.84876(5.99e-03)	0.91752(7.61e-03)	0.96375(6.45e-03)	0.98848(3.25e-03)	
`ren` to a name of a node at the same nesting depth +- 1	0.84874(5.88e-03)	0.91749(7.42e-03)	0.96420(6.41e-03)	0.98856(3.23e-03)	
Move 1 tree (total weight 1–500)	0.22297(4.42e-02)	0.29618(4.99e-02)	0.35883(5.11e-02)	0.41165(5.38e-02)	0.45939(4.81e-02)
Move 2 trees (total weight 2–500)	0.12542(4.03e-02)	0.16922(4.55e-02)	0.21187(5.53e-02)	0.25717(5.46e-02)	0.29564(5.91e-02)
Move 3 trees (total weight 3–500)	0.07630(2.49e-02)	0.10425(2.93e-02)	0.12864(3.42e-02)	0.15775(3.36e-02)	0.18557(3.31e-02)
Move 4 trees (total weight 4–500)	0.04970(1.33e-02)	0.06672(1.66e-02)	0.08891(1.86e-02)	0.10928(2.24e-02)	0.12919(2.47e-02)
`repN` with a randomly generated tree of a similar weight	0.34815(1.45e-02)	0.42905(1.59e-02)	0.49813(1.61e-02)	0.55772(1.60e-02)	0.60926(1.54e-02)
`repN` with 1 randomly generated tree weighting 1–10	0.35298(1.48e-02)	0.43261(1.54e-02)	0.50020(1.50e-02)	0.55768(1.49e-02)	0.60622(1.42e-02)
`repN` with 1 randomly generated tree weighting 10–25	0.33867(1.52e-02)	0.41159(1.61e-02)	0.47201(1.64e-02)	0.52354(1.59e-02)	0.56751(1.56e-02)
`repN` with 1 randomly generated tree weighting 25–50	0.33911(1.56e-02)	0.40956(1.66e-02)	0.46754(1.65e-02)	0.51616(1.65e-02)	0.55774(1.60e-02)
`repN` with 1 randomly generated tree weighting 50–500	0.33457(1.57e-02)	0.40207(1.67e-02)	0.45731(1.75e-02)	0.50321(1.71e-02)	0.54160(1.72e-02)
`repN` with 2 randomly generated trees (total weight 50–100)	0.33675(1.55e-02)	0.40540(1.65e-02)	0.46160(1.68e-02)	0.50868(1.72e-02)	0.54855(1.70e-02)
`repN` with 3 randomly generated trees (total weight 75–150)	0.33640(1.53e-02)	0.40413(1.67e-02)	0.46019(1.71e-02)	0.50673(1.72e-02)	0.54615(1.67e-02)
`repN` with a tree similar to one at the same nesting depth	0.27282(1.83e-02)	0.34084(1.95e-02)	0.39432(2.49e-02)	0.44392(2.73e-02)	0.48466(3.21e-02)
`repN` with a tree similar to one of its siblings	0.26829(2.08e-02)	0.33420(2.36e-02)	0.39096(2.79e-02)	0.44133(2.94e-02)	0.48443(3.03e-02)
`repN` with 1 real tree	0.22367(1.10e-02)	0.28181(1.33e-02)	0.33399(1.48e-02)	0.38269(1.76e-02)	0.42749(1.78e-02)
`repN` with 2 real trees	0.22189(8.29e-03)	0.27708(1.01e-02)	0.32588(1.13e-02)	0.37037(1.23e-02)	0.41073(1.38e-02)
`repN` with 3 real trees	0.22271(6.83e-03)	0.27781(8.36e-03)	0.32637(8.54e-03)	0.36910(9.40e-03)	0.40876(9.78e-03)
`ins`$^{\rightarrow}$ 1 randomly generated tree (weight similar to a sibling)	0.37017(2.20e-02)	0.44441(2.39e-02)	0.50467(2.52e-02)	0.55494(2.57e-02)	0.59686(2.65e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 1–10	0.37623(1.80e-02)	0.45076(1.93e-02)	0.51136(1.95e-02)	0.56147(1.96e-02)	0.60377(1.95e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 10–25	0.34611(1.67e-02)	0.41571(1.80e-02)	0.47244(1.88e-02)	0.52000(1.88e-02)	0.56017(1.88e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 25–50	0.34282(1.66e-02)	0.41113(1.78e-02)	0.46704(1.82e-02)	0.51336(1.83e-02)	0.55300(1.82e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 50–500	0.33493(1.59e-02)	0.40191(1.70e-02)	0.45675(1.73e-02)	0.50251(1.76e-02)	0.54083(1.77e-02)
`ins`$^{\rightarrow}$ 2 randomly generated trees (total weight 50-100)	0.33833(1.58e-02)	0.40598(1.71e-02)	0.46103(1.78e-02)	0.50679(1.80e-02)	0.54596(1.77e-02)
`ins`$^{\rightarrow}$ 3 randomly generated trees (total weight 75-150)	0.33763(1.58e-02)	0.40477(1.73e-02)	0.45957(1.73e-02)	0.50537(1.74e-02)	0.54415(1.75e-02)
`ins`$^{\rightarrow}$ 1 tree similar to one at the same nesting depth	0.29135(1.94e-02)	0.35328(2.04e-02)	0.40411(2.32e-02)	0.44722(2.57e-02)	0.48604(2.45e-02)
`ins`$^{\rightarrow}$ 1 tree similar to one of its siblings	0.28829(2.02e-02)	0.34982(2.32e-02)	0.40118(2.60e-02)	0.44405(2.73e-02)	0.48178(2.85e-02)
`ins`$^{\rightarrow}$ 1 real tree	0.23280(4.28e-03)	0.28743(5.46e-03)	0.33456(6.37e-03)	0.37515(6.82e-03)	0.41153(7.48e-03)
`ins`$^{\rightarrow}$ 2 real trees	0.23305(3.16e-03)	0.28786(3.77e-03)	0.33536(4.38e-03)	0.37658(4.69e-03)	0.41358(5.22e-03)
`ins`$^{\rightarrow}$ 3 real trees	0.23298(2.87e-03)	0.28777(3.35e-03)	0.33521(3.68e-03)	0.37681(3.86e-03)	0.41353(4.01e-03)
`del` a node weighting 1-500	0.33967(1.70e-02)	0.42840(1.80e-02)	0.50627(1.97e-02)	0.57778(1.79e-02)	0.63974(1.92e-02)
`del` a node weighting 1-10	0.36996(2.56e-02)	0.45844(2.77e-02)	0.53499(2.77e-02)	0.60248(2.69e-02)	0.66210(2.49e-02)
`del` a node weighting 10-25	0.29869(8.60e-03)	0.38670(7.32e-03)	0.47362(9.14e-03)	0.56707(1.25e-02)	0.66594(1.96e-03)
`del` a node weighting 25-500	0.29348(8.27e-03)	0.38246(1.26e-02)	0.47224(1.52e-02)	0.55574(2.07e-02)	0.64196(2.36e-02)
Random	0.33736(1.72e-02)	0.40553(1.82e-02)	0.46098(1.97e-02)	0.50724(1.87e-02)	0.54687(1.91e-02)
`repV` with a value already used in the document	0.81090(1.12e-02)	0.89777(4.25e-03)	1.02913(3.41e-02)		
`repV` with a randomly generated value	0.82164(1.03e-02)	0.91058(4.61e-03)	1.04513(3.64e-02)		
`repV` with that of a node with the same name	0.66485(5.60e-02)	0.71073(5.50e-02)	0.79642(3.97e-02)		

Table A.2: *pq*-gram based change ratio estimation (part 2).

Operation type	**0.01**	**0.05**	**0.1**	**0.15**	**0.2**
`ren` to a name already used in the document	0.02703(2.21e-03)	0.11585(7.70e-03)	0.19953(1.40e-02)	0.26053(2.12e-02)	0.30689(2.74e-02)
`ren` to a randomly generated name	0.02962(1.83e-03)	0.13001(3.73e-03)	0.22504(3.86e-03)	0.29661(2.97e-03)	0.35148(2.64e-03)
`ren` to a name of a node at the same nesting depth	0.02552(2.73e-03)	0.10607(9.85e-03)	0.18145(1.84e-02)	0.23693(2.57e-02)	0.27769(3.19e-02)
`ren` to a name of a node with the same parent name	0.02535(2.55e-03)	0.10634(1.05e-02)	0.18186(1.85e-02)	0.23696(2.62e-02)	0.27800(3.18e-02)
`ren` to a name of a node at the same nesting depth +- 1	0.02586(2.67e-03)	0.10876(9.28e-03)	0.18630(1.75e-02)	0.24373(2.38e-02)	0.28632(2.89e-02)
Move 1 tree (total weight 1–500)	0.00972(1.52e-03)	0.04567(4.01e-03)	0.08798(6.43e-03)	0.12969(6.37e-03)	0.16750(6.98e-03)
Move 2 trees (total weight 2–500)	0.00837(1.46e-03)	0.04059(4.87e-03)	0.07798(8.30e-03)	0.11445(1.03e-02)	0.14729(1.18e-02)
Move 3 trees (total weight 3–500)	0.00736(2.21e-03)	0.03718(7.27e-03)	0.07522(1.17e-02)	0.11132(1.10e-02)	0.14386(1.50e-02)
Move 4 trees (total weight 4–500)	0.00741(2.50e-03)	0.03951(6.47e-03)	0.07601(1.18e-02)	0.11162(1.29e-02)	0.14442(1.61e-02)
`repN` with a randomly generated tree of a similar weight	0.01283(7.74e-04)	0.06049(2.86e-03)	0.11565(4.92e-03)	0.16632(6.58e-03)	0.21296(7.72e-03)
`repN` with 1 randomly generated tree weighting 1–10	0.01296(8.84e-04)	0.06170(3.48e-03)	0.11752(5.49e-03)	0.16932(7.37e-03)	0.21667(8.66e-03)
`repN` with 1 randomly generated tree weighting 10–25	0.01332(7.00e-04)	0.06259(3.04e-03)	0.11838(5.11e-03)	0.16862(6.61e-03)	0.21404(7.78e-03)
`repN` with 1 randomly generated tree weighting 25–50	0.01368(7.17e-04)	0.06393(2.77e-03)	0.12062(4.80e-03)	0.17119(6.25e-03)	0.21660(7.60e-03)
`repN` with 1 randomly generated tree weighting 50–500	0.01372(6.13e-04)	0.06468(2.64e-03)	0.12154(4.55e-03)	0.17200(5.57e-03)	0.21700(6.84e-03)
`repN` with 2 randomly generated trees (total weight 50–100)	0.01374(5.94e-04)	0.06442(2.58e-03)	0.12116(4.69e-03)	0.17158(5.80e-03)	0.21669(6.98e-03)
`repN` with 3 randomly generated trees (total weight 75–150)	0.01374(5.90e-04)	0.06461(2.51e-03)	0.12145(4.41e-03)	0.17186(5.73e-03)	0.21706(6.92e-03)
`repN` with a tree similar to one at the same nesting depth	0.01016(9.03e-04)	0.04487(2.55e-03)	0.08287(5.06e-03)	0.11792(6.97e-03)	0.15017(9.00e-03)
`repN` with a tree similar to one of its siblings	0.00952(1.13e-03)	0.04154(3.36e-03)	0.07743(7.04e-03)	0.11200(8.36e-03)	0.14193(9.64e-03)
`repN` with 1 real tree	0.00822(9.79e-04)	0.03657(4.41e-03)	0.06778(9.24e-03)	0.09607(1.44e-02)	0.12228(1.80e-02)
`repN` with 2 real trees	0.00837(8.06e-04)	0.03850(3.84e-03)	0.07079(6.32e-03)	0.10313(1.04e-02)	0.12854(1.17e-02)
`repN` with 3 real trees	0.00872(5.65e-04)	0.04038(3.23e-03)	0.07516(5.68e-03)	0.10802(8.20e-03)	0.13691(9.67e-03)
`ins`$^{\rightarrow}$ 1 randomly generated tree (weight similar to a sibling)	0.01463(7.42e-04)	0.06859(3.47e-03)	0.12891(6.03e-03)	0.18249(8.39e-03)	0.23062(1.00e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 1–10	0.01470(7.66e-04)	0.06971(3.28e-03)	0.13087(5.74e-03)	0.18519(7.64e-03)	0.23361(9.08e-03)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 10–25	0.01395(6.80e-04)	0.06602(2.88e-03)	0.12414(4.87e-03)	0.17567(6.50e-03)	0.22176(7.63e-03)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 25–50	0.01395(6.55e-04)	0.06600(2.60e-03)	0.12385(4.64e-03)	0.17523(6.24e-03)	0.22080(7.27e-03)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 50–500	0.01392(5.96e-04)	0.06514(2.68e-03)	0.12212(4.29e-03)	0.17280(5.74e-03)	0.21803(6.73e-03)
`ins`$^{\rightarrow}$ 2 randomly generated trees (total weight 50-100)	0.01384(6.42e-04)	0.06540(2.60e-03)	0.12289(4.43e-03)	0.17374(5.83e-03)	0.21900(6.91e-03)
`ins`$^{\rightarrow}$ 3 randomly generated trees (total weight 75-150)	0.01393(5.43e-04)	0.06529(2.53e-03)	0.12283(4.37e-03)	0.17336(5.62e-03)	0.21894(6.49e-03)
`ins`$^{\rightarrow}$ 1 tree similar to one at the same nesting depth	0.01201(8.83e-04)	0.05402(2.57e-03)	0.10068(4.23e-03)	0.14195(5.33e-03)	0.17957(6.29e-03)
`ins`$^{\rightarrow}$ 1 tree similar to one of its siblings	0.01149(9.01e-04)	0.05237(2.80e-03)	0.09776(4.83e-03)	0.13829(6.00e-03)	0.17577(7.73e-03)
`ins`$^{\rightarrow}$ 1 real tree	0.00990(2.51e-04)	0.04716(6.80e-04)	0.08979(1.35e-03)	0.12853(1.85e-03)	0.16400(2.26e-03)
`ins`$^{\rightarrow}$ 2 real trees	0.00991(2.43e-04)	0.04755(6.66e-04)	0.09065(1.11e-03)	0.12972(1.25e-03)	0.16595(1.69e-03)
`ins`$^{\rightarrow}$ 3 real trees	0.00996(2.27e-04)	0.04768(7.58e-04)	0.09098(9.95e-04)	0.13017(1.43e-03)	0.16637(1.44e-03)
`del` a node weighting 1-500	0.01182(8.47e-04)	0.05653(2.51e-03)	0.10930(3.81e-03)	0.15914(5.43e-03)	0.20735(6.52e-03)
`del` a node weighting 1-10	0.01271(9.34e-04)	0.06082(3.66e-03)	0.11657(6.54e-03)	0.16925(8.55e-03)	0.21842(1.01e-02)
`del` a node weighting 10-25	0.01040(7.90e-04)	0.05084(2.51e-03)	0.10000(4.06e-03)	0.14778(4.12e-03)	0.19485(4.60e-03)
`del` a node weighting 25-500	0.00989(5.63e-04)	0.04908(1.38e-03)	0.09771(1.99e-03)	0.14565(3.13e-03)	0.19442(5.23e-03)
Random	0.01396(1.33e-03)	0.06478(3.22e-03)	0.12153(4.88e-03)	0.17187(6.10e-03)	0.21710(7.09e-03)
`repV` with a value already used in the document	0.03269(1.78e-03)	0.13482(6.87e-03)	0.22861(1.01e-02)	0.29964(1.33e-02)	0.35422(1.81e-02)
`repV` with a randomly generated value	0.03500(1.26e-03)	0.15482(3.66e-03)	0.27196(6.28e-03)	0.36428(1.03e-02)	0.43948(1.53e-02)
`repV` with that of a node with the same name	0.03253(2.19e-03)	0.13137(1.22e-02)	0.21927(2.00e-02)	0.28385(2.52e-02)	0.33217(2.86e-02)

Table A.3: *pql*-gram based change ratio estimation (part 1).

Operation type	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**
`ren` to a name already used in the document	0.36634(3.90e-02)	0.39975(4.91e-02)	0.41873(5.70e-02)	0.46175(4.97e-02)	
`ren` to a randomly generated name	0.42699(4.78e-03)	0.47521(5.96e-03)	0.50900(5.40e-03)	0.56874(1.21e-02)	
`ren` to a name of a node at the same nesting depth	0.32971(4.18e-02)	0.35754(4.93e-02)	0.37213(5.43e-02)	0.40802(4.62e-02)	
`ren` to a name of a node with the same parent name	0.33005(4.21e-02)	0.35764(4.96e-02)	0.37209(5.43e-02)	0.40840(4.61e-02)	
`ren` to a name of a node at the same nesting depth +- 1	0.34226(3.80e-02)	0.37371(4.35e-02)	0.39164(4.68e-02)	0.43161(3.69e-02)	
Move 1 tree (total weight 1–500)	0.23706(8.03e-03)	0.29957(9.02e-03)	0.35228(1.06e-02)	0.39792(1.43e-02)	0.44107(2.23e-02)
Move 2 trees (total weight 2–500)	0.21178(1.23e-02)	0.27167(1.17e-02)	0.32201(1.53e-02)	0.37107(1.55e-02)	0.41539(1.68e-02)
Move 3 trees (total weight 3–500)	0.20611(1.66e-02)	0.25871(2.26e-02)	0.31484(2.01e-02)	0.36302(2.01e-02)	0.40643(2.06e-02)
Move 4 trees (total weight 4–500)	0.20879(2.11e-02)	0.26571(2.13e-02)	0.31650(2.16e-02)	0.36709(1.99e-02)	0.41515(2.09e-02)
`repN` with a randomly generated tree of a similar weight	0.29697(9.03e-03)	0.36985(9.51e-03)	0.43328(9.36e-03)	0.48912(8.74e-03)	0.53827(8.04e-03)
`repN` with 1 randomly generated tree weighting 1–10	0.30137(9.32e-03)	0.37368(9.35e-03)	0.43628(8.95e-03)	0.49091(8.56e-03)	0.53811(7.79e-03)
`repN` with 1 randomly generated tree weighting 10–25	0.29349(8.93e-03)	0.36038(9.55e-03)	0.41740(9.49e-03)	0.46659(9.32e-03)	0.50931(9.21e-03)
`repN` with 1 randomly generated tree weighting 25–50	0.29483(8.67e-03)	0.36021(9.43e-03)	0.41517(9.69e-03)	0.46226(9.65e-03)	0.50312(9.45e-03)
`repN` with 1 randomly generated tree weighting 50–500	0.29377(8.10e-03)	0.35719(8.89e-03)	0.41021(9.57e-03)	0.45536(9.36e-03)	0.49376(9.58e-03)
`repN` with 2 randomly generated trees (total weight 50–100)	0.29405(8.29e-03)	0.35821(8.95e-03)	0.41208(9.25e-03)	0.45784(9.64e-03)	0.49739(9.59e-03)
`repN` with 3 randomly generated trees (total weight 75–150)	0.29445(8.00e-03)	0.35790(8.96e-03)	0.41153(9.29e-03)	0.45704(9.46e-03)	0.49607(9.23e-03)
`repN` with a tree similar to one at the same nesting depth	0.20815(1.18e-02)	0.25961(1.30e-02)	0.30204(1.50e-02)	0.34251(1.70e-02)	0.37776(1.75e-02)
`repN` with a tree similar to one of its siblings	0.19952(1.34e-02)	0.24990(1.49e-02)	0.29328(1.74e-02)	0.33361(1.78e-02)	0.36927(1.78e-02)
`repN` with 1 real tree	0.16832(2.41e-02)	0.21269(3.25e-02)	0.25065(3.85e-02)	0.28616(4.43e-02)	0.32025(5.12e-02)
`repN` with 2 real trees	0.17891(1.65e-02)	0.22290(2.09e-02)	0.26299(2.29e-02)	0.29942(2.67e-02)	0.33276(2.88e-02)
`repN` with 3 real trees	0.18984(1.32e-02)	0.23700(1.57e-02)	0.27815(1.58e-02)	0.31461(1.80e-02)	0.34966(1.93e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree (weight similar to a sibling)	0.31225(1.30e-02)	0.37998(1.45e-02)	0.43654(1.57e-02)	0.48467(1.63e-02)	0.52575(1.71e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 1–10	0.31615(1.11e-02)	0.38418(1.23e-02)	0.44108(1.28e-02)	0.48930(1.32e-02)	0.53062(1.34e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 10–25	0.30040(9.48e-03)	0.36524(1.05e-02)	0.41943(1.12e-02)	0.46563(1.15e-02)	0.50536(1.16e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 25–50	0.29878(9.03e-03)	0.36281(9.91e-03)	0.41647(1.03e-02)	0.46189(1.06e-02)	0.50114(1.08e-02)
`ins`$^{\rightarrow}$ 1 randomly generated tree weighting 50–500	0.29446(8.32e-03)	0.35775(9.07e-03)	0.41064(9.37e-03)	0.45550(9.80e-03)	0.49387(1.00e-02)
`ins`$^{\rightarrow}$ 2 randomly generated trees (total weight 50-100)	0.29634(8.38e-03)	0.36000(9.24e-03)	0.41307(9.90e-03)	0.45801(1.02e-02)	0.49693(1.01e-02)
`ins`$^{\rightarrow}$ 3 randomly generated trees (total weight 75-150)	0.29610(8.28e-03)	0.35936(9.34e-03)	0.41228(9.44e-03)	0.45735(9.63e-03)	0.49596(9.85e-03)
`ins`$^{\rightarrow}$ 1 tree similar to one at the same nesting depth	0.24477(7.99e-03)	0.29925(8.72e-03)	0.34641(8.69e-03)	0.38724(9.28e-03)	0.42305(8.76e-03)
`ins`$^{\rightarrow}$ 1 tree similar to one of its siblings	0.24058(8.20e-03)	0.29555(8.98e-03)	0.34243(9.75e-03)	0.38279(9.73e-03)	0.41901(9.70e-03)
`ins`$^{\rightarrow}$ 1 real tree	0.22685(3.14e-03)	0.28107(4.03e-03)	0.32806(4.65e-03)	0.36900(4.95e-03)	0.40546(5.34e-03)
`ins`$^{\rightarrow}$ 2 real trees	0.22938(2.09e-03)	0.28395(2.51e-03)	0.33130(2.91e-03)	0.37264(3.06e-03)	0.40945(3.45e-03)
`ins`$^{\rightarrow}$ 3 real trees	0.23023(1.81e-03)	0.28496(2.20e-03)	0.33232(2.39e-03)	0.37395(2.55e-03)	0.41063(2.69e-03)
`del` a node weighting 1-500	0.29671(7.92e-03)	0.37940(8.10e-03)	0.45514(1.00e-02)	0.52617(9.78e-03)	0.59011(1.11e-02)
`del` a node weighting 1-10	0.30829(1.20e-02)	0.38909(1.35e-02)	0.46257(1.37e-02)	0.52964(1.37e-02)	0.59083(1.36e-02)
`del` a node weighting 10-25	0.28568(4.96e-03)	0.37394(9.70e-03)	0.46105(1.58e-02)	0.55907(1.93e-02)	0.66354(7.79e-04)
`del` a node weighting 25-500	0.29032(7.75e-03)	0.37949(1.20e-02)	0.46930(1.48e-02)	0.55299(2.03e-02)	0.63935(2.35e-02)
Random	0.29402(8.51e-03)	0.35766(9.33e-03)	0.41103(9.81e-03)	0.45633(9.86e-03)	0.49566(9.84e-03)
`repV` with a value already used in the document	0.43111(3.02e-02)	0.47940(4.54e-02)	0.53043(6.53e-02)		
`repV` with a randomly generated value	0.55525(2.55e-02)	0.64054(3.54e-02)	0.73944(5.86e-02)		
`repV` with that of a node with the same name	0.39680(3.23e-02)	0.43376(3.50e-02)	0.47774(3.23e-02)		

Table A.4: *pql*-gram based change ratio estimation (part 2).

Appendix B

Extended Discussion for OA4QA Track

Section B.1 provides the necessary details about the dataset and the query evaluation engine. Section B.2 illustrates the general results for the track and correlates them with the results for *conference*. Section B.3 shows all the test queries, while Section B.4 provides detailed results for the *OA4QA* evaluation.

B.1 Dataset and Query Evaluation

The set of ontologies coincides with that of the *conference* track of *OAEI*, in order to facilitate the understanding of the queries and query results. The dataset is however extended with synthetic ABoxes, extracted from the *DBLP* dataset.[1]

Given a query q expressed using the vocabulary of ontology $\mathcal{O}_1$, another ontology $\mathcal{O}_2$ enriched with synthetic data is chosen. Finally, the query is executed over the aligned ontology $\mathcal{O}^{\mathcal{M}} = \mathcal{O}_1 \cup \mathcal{M} \cup \mathcal{O}_2$, where $\mathcal{M}$ is an alignment between $\mathcal{O}_1$ and $\mathcal{O}_2$.

The query evaluation engine we use is an extension of the OWL 2 reasoner HermiT, known as OWL-BGP[2] [KGH11]. OWL-BGP is able to process SPARQL queries in the SPARQL-OWL fragment, under the OWL 2 Direct Semantics entailment regime [KGH11]. The queries employed in the *OA4QA* track are standard conjunctive queries, that are fully supported by the more expressive SPARQL-OWL fragment. SPARQL-OWL, for instance, also support queries where variables occur within complex class expressions or bind to class or property names.

The evaluation metrics used for the *OA4QA* track are the classic information retrieval ones (*i.e.*, precision, recall and f-measure), but on the result set of the query evaluation.

[1] http://dblp.uni-trier.de/xml/

[2] https://code.google.com/p/owl-bgp/

The aforementioned metrics are then evaluated, for each alignment computed by the different matching tools, against the publicly available reference alignments *ra1*, and its manually repaired version of *ra1* from conservativity and consistency violations, called *rar1* (not to be confused with *ra2* alignment of the *conference* track, that is not publicly available).

Three categories of queries are considered in *OA4QA* (the full list of queries is available in Section B.3): *(i)* basic queries: instance retrieval queries for a single class or queries involving at most one trivial correspondence (that is, correspondences between entities with (quasi-)identical names), *(ii)* queries involving (consistency or conservativity) violations, *(iii)* advanced queries involving nontrivial correspondences.

For unsatisfiable ontologies, an additional repair step is attempted, consisting in the removal of all the individuals of incoherent classes. In some cases, this allowed to answer the query, and depending on the classes involved in the query itself, sometimes it did not interfere in the query answering process.

B.2 Track Results

Table B.1 shows the average precision, recall and f-measure results for the whole set of queries: AML, LogMap, LogMapC and XMap were the only matchers whose alignments allowed to answer all the queries of the evaluation.

Considering Table B.1, the difference in results between the publicly available reference alignment of the *conference* track (*ra1*) and its repaired version (*rar1*) is not significant and, as expected, affects precision. Most of the differences between *ra1* and *rar1* are related to conservativity violations, and this is reflected by a reduced precision employing *rar1* w.r.t. *ra1*. However, the f-measure ranking between the two reference alignments is (mostly) preserved. If we compare Table B.1 (the results of the *OA4QA* track) and Table B.3 (the results of *conference* track) we can see that the top-4 matcher ranking coincides, even if with a slight variation. But, considering *rar1* alignment, the gap between the top-4 matcher and the others is highlighted, and it also allows to differentiate more among the least performing matchers, and seems therefore more suitable as a reference alignment for evaluating matchers in advanced tasks (such as, query answering).

Comparing Table B.1 and Table B.2 (measuring the degree of incoherence of the computed alignments of the *conference* track) it seems that a negative correlation between the ability of answering queries and the average degree of incoherence of the matchers exists. For instance, taking into account the different positions in the ranking of AOT, we can see that logical violations are definitely penalized more in our test case than in the traditional *conference* track, due to its target scenario. MaasMatch, instead, even if scoring poorly w.r.t. *ra1*, and despite the high number of incoherences, is in general able to answer enough of the test queries (5 out of 18).

Concerning the conservativity principle, a relevant number of violations (fourth column of Table B.2) on the system performance is present, but the results are more complicated and calls for additional comments.

The worse-performing matchers (namely, AOTL, AOT and MaasMatch), suffer from a high number of conservativity principle violations, as expected. LogMapLt, despite a relatively low violation rate, has been penalized by its incoherences, that prevented to answer 7 out of 18 queries.

OMReasoner and RSDLWB, instead, have also been penalized by the small size of the computed alignments (*i.e.*, they were missing mappings needed to answer some of the proposed queries).

The top-4 matching systems produce only incoherences-free alignments, and therefore the different performance in this track is only influenced by the presence/absence of a mapping required by a query and conservativity violations.

XMap, despite having the least number of conservativity violations and producing incoherences-free alignments, has been heavily penalized by many missing mappings (among those required by advanced queries in particular, see Table B.11) and low recall (*i.e.*, "wrong" mappings that contributed to queries with incorrect results).

AML computes the best approximation of the reference alignment in *conference* track, but the presence of conservativity violations affects its performance in query answering, voiding the gap between AML and LogMapC, and allowing LogMap to score best in *OA4QA*.

LogMapC deserves further comments because, to the best of our knowledge, is the only ontology matching systems addressing conservativity principle violations (it uses the repair technique presented in Chapter 6). LogMapC does not outperform LogMap, because some mappings removed by its extended repair capabilities prevents to answer to query q-$v1$ of Listings B.7, Section B.3, (the result set is empty as an effect of mapping removal, as can be seen from Table B.6, Table B.9 and Table B.12. Instead, query q-$v6$ (Listings B.12), involves a conservativity violation affecting the alignment computed by LogMap, but not LogMapC, thanks to its extended repair capabilities, for which LogMapC has recall equal to 1, while LogMap scores 0.666.

Of course the number of queries and violations cannot be considered statistically significant (*i.e.*, with an extremely low bias), and for this reason it does not allow to highlight the existing correlations as much as desired. Notable exceptions are AOT, AOTL and MaasMatch matchers, for which the role of logical violations is quite evident (favored by their massive number, as shown in Table B.2). These limitations will be addressed by further explorations in future editions of the evaluation.

Matcher	Answered queries	ra1			rar1		
		Prec.	F-m	Rec.	Prec.	F-m	Rec.
LogMap	18/18	0.750	0.750	0.750	0.729	0.739	0.750
AML	18/18	0.722	0.708	0.694	0.701	0.697	0.694
LogMapC	18/18	0.722	0.708	0.694	0.722	0.708	0.694
XMap	18/18	0.556	0.519	0.487	0.554	0.518	0.487
RSDLWB	15/18	0.464	0.471	0.479	0.407	0.431	0.458
OMReasoner	15/18	0.409	0.432	0.458	0.407	0.431	0.458
LogMapLt	11/18	0.409	0.416	0.423	0.351	0.375	0.402
MaasMatch	5/18	0.223	0.247	0.278	0.203	0.235	0.278
AOTL	6/18	0.056	0.056	0.056	0.056	0.056	0.056
AOT	0/18	0.000	0.000	0.000	0.000	0.000	0.000

Table B.1: *OA4QA* track, averaged precision and recall (over the single queries), for each matcher. F-measure, instead, is computed using the averaged precision and recall. Matchers are sorted on their f-measure values for *ra1*.

Matcher	Alignment Size	Inc. Alignments	Inc. Degree	Total diff$^{\approx}$
AML	10.95	0/21	0%	47
AOT	59.17	18/21	40.4%	$\geq$2099
AOTL	14.67	17/21	15.1%	1392
LogMap	10.71	0/21	0%	36
LogMapC	10.24	0/21	0%	5
LogMapLt	9.91	7/21	5.4%	12
MaasMatch	33.00	19/21	21%	700
OMReasoner	8.10	4/21	2.5%	6
RSDLWB	8.33	4/21	2.5%	7
XMap	8.14	0/21	0%	2

Table B.2: Incoherences in the alignments computed by the participants to the Conference track. The values in the "Alignment size" and "Inc. Degre" columns represent averages over the 21 computed alignments. The last column is the sum of the conservativity violations for all the alignment of the given matcher.

Matcher	Precision	Recall	F-Measure
AML	0.85	0.64	0.73
LogMap	0.8	0.59	0.68
LogMapC	0.82	0.57	0.67
XMap	0.87	0.49	0.63
AOT	0.8	0.47	0.59
RSDLWB	0.81	0.47	0.59
LogMapLt	0.73	0.5	0.59
OMReasoner	0.82	0.46	0.59
AOTL	0.77	0.43	0.55
MaasMatch	0.64	0.48	0.55

Table B.3: Conference track results, matchers are sorted on their f-measure values.

B.3 Test Queries

In this section the full list of test queries for *OA4QA* track is given, for each query kind. For brevity we omit the following set of prefixes from the begin of each query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX cmt: <http://cmt#>
PREFIX conference: <http://conference#>
PREFIX confof: <http://confOf#>
PREFIX edas: <http://edas#>
PREFIX ekaw: <http://ekaw#>
PREFIX iasted: <http://iasted#>
PREFIX sigkdd: <http://sigkdd#>
PREFIX foaf: <http://xmlsn.com/foaf/0.1#>
```

Basic Queries.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type cmt:Author.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.1: Basic query 1, $\mathcal{O}_{cmt}$ over $\mathcal{O}_{ekaw}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type conference:Conference.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.2: Basic query 2, $\mathcal{O}_{cmt}$ over $\mathcal{O}_{conference}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type edas:Review.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.3: Basic query 3, $\mathcal{O}_{edas}$ over $\mathcal{O}_{ekaw}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type iasted:City.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.4: Basic query 4, $\mathcal{O}_{iasted}$ over $\mathcal{O}_{confOf}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type sigkdd:Program_Committee.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.5: Basic query 5, $\mathcal{O}_{sigkdd}$ over $\mathcal{O}_{conference}$.

```
SELECT DISTINCT ?x
WHERE {
  <http://xmlns.com/foaf/0.1#EmilioMancini> rdf:type cmt:Author.
  <http://xmlns.com/foaf/0.1#EmilioMancini> rdf:type owl:NamedIndividual.
  ?x cmt:hasAuthor <http://xmlns.com/foaf/0.1#EmilioMancini>.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.6: Basic query 6, $\mathcal{O}_{cmt}$ over $\mathcal{O}_{edas}$.

Violation Queries.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type ekaw:Conference_Participant.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.7: Violation query 1, $\mathcal{O}_{ekaw}$ over $\mathcal{O}_{confOf}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type iasted:Speaker .
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.8: Violation query 2, $\mathcal{O}_{iasted}$ over $\mathcal{O}_{sigkdd}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type iasted:Author.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.9: Violation query 3, $\mathcal{O}_{iasted}$ over $\mathcal{O}_{sigkdd}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type confof:Author.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.10: Violation query 4, $\mathcal{O}_{confOf}$ over $\mathcal{O}_{ekaw}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type sigkdd:Speaker.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.11: Violation query 5, $\mathcal{O}_{conference}$ over $\mathcal{O}_{sigkdd}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type confof:Participant.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.12: Violation query 6, $\mathcal{O}_{confOf}$ over $\mathcal{O}_{conference}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type confof:Participant.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.13: Violation query 7, $\mathcal{O}_{confOf}$ over $\mathcal{O}_{ekaw}$.

Advanced Queries.

```
SELECT DISTINCT ?x
WHERE {
 ?x rdf:type [
   a owl:Class;
    owl:unionOf (
          confof:Trip
          confof:Banquet
          confof:Reception
    )
   ] .
  ?x rdf:type owl:NamedIndividual.
```

```
}
```

Listing B.14: Advanced query 1, $\mathcal{O}_{confOf}$ over $\mathcal{O}_{edas}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type confof:Chair_PC.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.15: Advanced query 2, $\mathcal{O}_{confOf}$ over $\mathcal{O}_{cmt}$.

```
SELECT DISTINCT ?x
WHERE {
  ?y rdf:type owl:NamedIndividual.
  ?x rdf:type owl:NamedIndividual.
  ?x ekaw:reviewWrittenBy ?y.
}
```

Listing B.16: Advanced query 3, $\mathcal{O}_{ekaw}$ over $\mathcal{O}_{cmt}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type ekaw:Paper_Author.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.17: Advanced query 4, $\mathcal{O}_{ekaw}$ over $\mathcal{O}_{conference}$.

```
SELECT DISTINCT ?x
WHERE {
  ?x rdf:type edas:Attendee.
  ?x rdf:type owl:NamedIndividual.
}
```

Listing B.18: Advanced query 5, $\mathcal{O}_{edas}$ over $\mathcal{O}_{confOf}$.

B.4 Detailed Results

Table B.4 Table B.5 Table B.6 provide precision detailed results for basic, advanced and violation queries, respectively.

Table B.7 Table B.8 Table B.9 provide recall detailed results for basic, advanced and violation queries, respectively.

Table B.10 Table B.11 Table B.12 provide f-measure detailed results for basic, advanced and violation queries, respectively.

Matcher	q-b1 (ra1)	q-b1 (rar1)	q-b2 (ra1)	q-b2 (rar1)	q-b3 (ra1)	q-b3 (rar1)	q-b4 (ra1)	q-b4 (rar1)	q-b5 (ra1)	q-b5 (rar1)	q-b6 (ra1)	q-b6 (rar1)
AML	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
LogMapC	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
LogMapLite	0.000	0.000	1.000	1.000	0.000	0.000	1.000	1.000	1.000	1.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
RSDLWB	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
XMap	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table B.4: Detailed precision results for *OA4QA* track (basic queries).

Matcher	q-a1 (ra1)	q-a1 (rar1)	q-a2 (ra1)	q-a2 (rar1)	q-a3 (ra1)	q-a3 (rar1)	q-a4 (ra1)	q-a4 (rar1)	q-a5 (ra1)	q-a5 (rar1)
AML	1.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	1.000	1.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
LogMapC	1.000	1.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
LogMapLite	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
RSDLWB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
XMap	1.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table B.5: Detailed precision results for *OA4QA* track (advanced queries).

Matcher	q-v1 (ra1)	q-v1 (rar1)	q-v2 (ra1)	q-v2 (rar1)	q-v3 (ra1)	q-v3 (rar1)	q-v4 (ra1)	q-v4 (rar1)	q-v5 (ra1)	q-v5 (rar1)	q-v6 (ra1)	q-v6 (rar1)	q-v7 (ra1)	q-v7 (rar1)
AML	1.000	0.629	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.991
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	1.000	0.629	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.499	0.499	1.000	0.991
LogMapC	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.991
LogMapLite	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	0.510	0.510	0.510	0.510	0.334	0.334	0.000	0.000	0.000	0.000	0.000	0.000
RSDLWB	1.000	0.000	0.510	0.510	0.510	0.510	0.334	0.334	0.000	0.000	0.000	0.000	0.000	0.000
XMap	0.000	0.000	1.000	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	0.000	1.000	0.967

Table B.6: Detailed precision results for *OA4QA* track (violation queries).

Matcher	q-b1 (ra1)	q-b1 (rar1)	q-b2 (ra1)	q-b2 (rar1)	q-b3 (ra1)	q-b3 (rar1)	q-b4 (ra1)	q-b4 (rar1)	q-b5 (ra1)	q-b5 (rar1)	q-b6 (ra1)	q-b6 (rar1)
AML	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
LogMapC	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
LogMapLite	0.000	0.000	1.000	1.000	0.000	0.000	1.000	1.000	1.000	1.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
RSDLWB	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
XMap	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table B.7: Detailed recall results for *OA4QA* track (basic queries).

Matcher	q-a1 (ra1)	q-a1 (rar1)	q-a2 (ra1)	q-a2 (rar1)	q-a3 (ra1)	q-a3 (rar1)	q-a4 (ra1)	q-a4 (rar1)	q-a5 (ra1)	q-a5 (rar1)
AML	0.500	0.500	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	0.500	0.500	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
LogMap	0.500	0.500	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
LogMapC	0.500	0.500	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
LogMapLite	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
RSDLWB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
XMap	0.500	0.500	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table B.8: Detailed recall results for *OA4QA* track (advanced queries).

Matcher	q-v1 (ra1)	q-v1 (rar1)	q-v2 (ra1)	q-v2 (rar1)	q-v3 (ra1)	q-v3 (rar1)	q-v4 (ra1)	q-v4 (rar1)	q-v5 (ra1)	q-v5 (rar1)	q-v6 (ra1)	q-v6 (rar1)	q-v7 (ra1)	q-v7 (rar1)
AML	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
LogMapC	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
LogMapLite	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	1.000	1.000	1.000	1.000	0.970	0.970	0.000	0.000	0.000	0.000	0.000	0.000
RSDLWB	0.371	0.000	1.000	1.000	1.000	1.000	0.970	0.970	0.000	0.000	0.000	0.000	0.000	0.000
XMap	0.000	0.000	1.000	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	0.000	0.273	0.266

Table B.9: Detailed recall results for *OA4QA* track (violation queries).

Matcher	q-b1 (ra1)	q-b1 (rar1)	q-b2 (ra1)	q-b2 (rar1)	q-b3 (ra1)	q-b3 (rar1)	q-b4 (ra1)	q-b4 (rar1)	q-b5 (ra1)	q-b5 (rar1)	q-b6 (ra1)	q-b6 (rar1)
AML	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
LogMapC	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
LogMapLite	0.000	0.000	1.000	1.000	0.000	0.000	1.000	1.000	1.000	1.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
RSDLWB	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
XMap	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table B.10: Detailed f-measure results for *OA4QA* track (basic queries).

Matcher	q-a1 (ra1)	q-a1 (rar1)	q-a2 (ra1)	q-a2 (rar1)	q-a3 (ra1)	q-a3 (rar1)	q-a4 (ra1)	q-a4 (rar1)	q-a5 (ra1)	q-a5 (rar1)
AML	0.667	0.667	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	0.667	0.667	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
LogMapC	0.667	0.667	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
LogMapLite	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
RSDLWB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
XMap	0.667	0.667	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table B.11: Detailed f-measure results for *OA4QA* track (advanced queries).

Matcher	q-v1 (ra1)	q-v1 (rar1)	q-v2 (ra1)	q-v2 (rar1)	q-v3 (ra1)	q-v3 (rar1)	q-v4 (ra1)	q-v4 (rar1)	q-v5 (ra1)	q-v5 (rar1)	q-v6 (ra1)	q-v6 (rar1)	q-v7 (ra1)	q-v7 (rar1)
AML	1.000	0.772	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.995
AOT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AOTL	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LogMap	1.000	0.772	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.666	0.666	1.000	0.995
LogMapC	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.995
LogMapLite	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MaasMatch	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
OMReasoner	0.000	0.000	0.676	0.676	0.676	0.676	0.497	0.497	0.000	0.000	0.000	0.000	0.000	0.000
RSDLWB	0.542	0.000	0.676	0.676	0.676	0.676	0.497	0.497	0.000	0.000	0.000	0.000	0.000	0.000
XMap	0.000	0.000	1.000	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	0.000	0.429	0.417

Table B.12: Detailed f-measure results for *OA4QA* track (violation queries).

Appendix C

Appendix for Equivalence Violations

C.1 Alternative *ASP* Program for Conservative Diagnoses

In this section we detail (Listing C.1) the *ASP* program that computes (one of) the best approximation for a conservative diagnosis based on lost equivalence entailments between entities of the two ontologies, in order to estimate the effect of the computed diagnosis.

C.2 CycleBreaker Temporal Complexity

In this section we analyze the temporal complexity of the two variants of the CycleBreaker algorithm, as well as the one of its relevant components. As a preliminar step, in Proposition C.1 we investigate the complexity of createDigraph function.

Proposition C.1. Given two ontologies $\mathcal{O}_1, \mathcal{O}_2$ and an alignment $\mathcal{M}$ between them, function createDigraph computes the associated digraph representation $G = (V, A)$ of $\mathcal{O}_1, \mathcal{O}_2$, and $\mathcal{M}$, with time complexity in $\mathcal{O}(m)$, where $m = |Axioms(\mathcal{O}_1)| + |Axioms(\mathcal{O}_2)| + |\mathcal{M}|$.

Proof. We start by analyzing the relationships among the digraph size, denoted with $n = |V| + |A|$, and the ontologies and alignment it represents. $|A| \leq m$, while $|V| \leq 2 \cdot (|Axioms(\mathcal{O}_1)| + |Axioms(\mathcal{O}_2)|)$. Given that $createDigraph(\mathcal{O}_1, \mathcal{O}_2, \mathcal{M})$ scans the sets $Axioms(\mathcal{O}_1)$, $Axioms(\mathcal{O}_2)$, $\mathcal{M}$ and given that, at each iteration, it performs constant time operations, its time complexity is in $\mathcal{O}(m)$, and therefore also in $\mathcal{O}(n)$. This concludes the proof.

□

```
% r0
#domain vtx(X,O).
#domain vtx(Y,P).
#domain vtx(Z,Q).

% r1
reachesPre(X,Y) :- edge(X,Y,C,M), X!=Y .
% r2
reachesPre(X,Z) :- reachesPre(X,Y), edge(Y,Z,C,M), X!=Y, Y!=Z .

% r3
reaches(X,Y) :- edge(X,Y,C,M), not_removed(edge(X,Y,C,M)), X!=Y.
% r4
reaches(X,Z) :- reaches(X,Y), edge(Y,Z,C,M), not_removed(edge(Y,Z,C,M)), X!=Y, Y!=Z .

% r5a
reachesSafe(X,X) .
% r5b
reachesSafe(X,Y) :- edge(X,Y,C,0), O=P, X!=Y.
% r6
reachesSafe(X,Z) :- reachesSafe(X,Y), edge(Y,Z,C,0), O=P, X!=Y, Y!=Z .

% r7
not_removed(edge(X,Y,C,M)) | removed(edge(X,Y,C,M)) :- edge(X,Y,C,M), X!=Y .

% r8
not_removed(edge(X,Y,C,0)) :- edge(X,Y,C,0), X!=Y, O=P.

% r9a
notEQ(X,Y) :- reachesSafe(X,A), reachesSafe(A,X), reachesSafe(B,Y), reachesSafe(Y,B),
    reachesPre(X,Y), not reaches(X,Y), edge(A,B,C,1), vtx(A,O), vtx(B,P), O!=P, X!=Y, A!=B,
    A!=Y, B!=X .
% r9b
#minimize [ notEQ(X,Y) = 1 @ 1 : X!=Y : O!=P ] .

% r10
unsafeCycle(Y) :- not reachesSafe(Y,X), reaches(Y,X), reaches(X,Y), O=P, X!=Y.

% r11
:- unsafeCycle(Y) .

% r12
#minimize [ removed(edge(X,Y,C,1)) = C @ 2] .

% r13
#hide .
#show removed/1.
```

Listing C.1: *ASP* program for computing an approximation of a minimal diagnosis for the *CMAP-WFES* problem based on lost equivalence entailments.

In Proposition C.2 we characterize the relationship between digraph size (in term of vertices and arcs) and mapping set cardinality, that will be useful in the remainder of the section.

Proposition C.2. Given a digraph $G = (V, A)$ and an alignment $\mathcal{M}$ such that $\mathcal{M} \subseteq A$, then $n > m$, where $n = |V| + |A|$, and $m = |\mathcal{M}|$.

Proof. From Proposition 5.14 we have that $|V| = \sqrt{\frac{m}{2}}$, while the lower bound for $|A|$ is m. Therefore, $n \geq m + \sqrt{\frac{m}{2}}$, and thus $n > m$ holds, proving the proposition.

□

In order to derive the complexity of Algorithm 13, we first need to analyze the temporal complexity of the different possible implementations for the solver function, that is, greedyDiagnosis function (Proposition C.3), the GA-based heuristic (Proposition C.4). For the *ASP*-based implementation we reuse the results given in Proposition 5.11.

Proposition C.3. Given an alignment $\mathcal{M}$, and the associated graph representation $G = (V, A)$, with $minCycles$ as the set of minimal cycles of G, the time complexity of greedyDiagnosis$(cycles, \mathcal{M})$ is in $\mathcal{O}(m \cdot (\log m + n))$, where $m = |\mathcal{M}|$, $cycles \subseteq minCycles$, and $|cycles| = c$.

Proof. We separately analyze the main subcomponents of the algorithm, and the analysis concludes by summing their contributes.

1. Time complexity for the operations of lines 4– 14 is equal to $c \cdot m \cdot c_1$:
   - using appropriate data structures (a hash-based set implementation for the set of intercepted cycles and a hash-map for $intercept$ map) the operations at lines 7–11 can be performed in constant time c_1,
   - given that each cycle may traverse all the mappings of $\mathcal{M}$, the loop of lines 5–13 may be executed at most m times,
2. $intercept$ has at most m elements (line 15), and the time complexity of sorting them is equal to $m \log m$ (using, for instance, *Merge Sort* algorithm), for each sorting condition (in our case there are 2), so totalling a time complexity equal to $2 \cdot m \log m$,
3. the loop statement of lines 16–23 is executed at most m times, with a total time complexity equal to $m \cdot (c \cdot c_2 + c_3 + c_4)$:
   - the set of cycles broken by the elements of a diagnosis is updated at each modification of the diagnosis, and the update requires constant time c_2 (*e.g.*, using an hash-set) the test of line 17 has, therefore, time complexity in $c \cdot c_2$, given that each mapping may be traversed by all the n cycles,

- diagnosis update of line 18 can be performed in constant time c_3 by implementing the diagnosis as an hash-set,
- test of line 20 can be efficiently realized by reducing it to a cardinality comparison requiring constant time c_4,

The total complexity is thus equal to $(2 \cdot m \log m) + (c \cdot m \cdot c_1) + (m \cdot (c \cdot c_2 + c_3 + c_4))$, that is in $\mathcal{O}(m \log m + c \cdot m)$, and therefore also in $\mathcal{O}(m \cdot (\log m + c))$. This concludes the proof.

□

In Proposition C.4, the temporal complexity of the GA-based heuristic is presented. Given that the worst-case needs to be investigated, we restrict our analysis to the situation in which the terminating condition is the only maximum number of iterations terminating strategy. Best global solution stagnation terminating condition, in fact, may only be met before reaching the maximum number of iterations, and therefore can never represent the worst-case for the considered heuristic.

Proposition C.4. Given an alignment $\mathcal{M}$ and the associated graph representation $G = (V, A)$ with a set of cycles to break, namely $cycles$, the temporal complexity of the GA-based heuristic is in $\mathcal{O}((m^2 \cdot (\log m + c) \cdot (p - 1)) + ((it - 1) \cdot (p - r) \cdot m \log m \cdot c))$, where:

- p is the population size,
- r the individuals that are retained from the previous generation,
- m is the cardinality of $\mathcal{M}$,
- it is the maximum number of iterations,
- c is the cardinality of $cycles$ set.

Proof. We start by analyzing the constant costs for useful subtasks used throughout the computation performed by the GA-based heuristic:

- c_1 is the cost of randomly choosing a binary value using a (pseudo-)random value generator,
- c_2 is the cost of mutating the *j-th* position of an individual, that is, complementing the binary value with a certain probability,
- c_3 is the cost of generating an individual using crossover from a pair of parents, that reduces at copying two substrings from the parents,

- c_4 is the cost of performing a bit-wise and operation (denoted as $\&$) between two integers (*i.e.*, encodings),
- c_5 is the cost of complementing at most m bits in the encoding of an individual,
- c_6 is the cost of retrieving the mapping associated to the *k-th* position of the encoding of an individual and extracting its weight; constant time is achieved using an hash-map between positions and associated mappings.

In what follows we analyze the runtime complexity of the different components of the GA:

1. Cost of computing the first generation: it consists on the complexity of invoking $\mathsf{greedyDiagnosis}(cycles, \mathcal{M})$ function, summed to the cost of generating the other $p-1$ individuals, that is $(p-1) \cdot m \cdot c_1$; the cost of computing the first generation is in $\mathcal{O}(m \cdot (\log m + c) + (p-1) \cdot m \cdot c_1)$, and therefore also in $\mathcal{O}(n^2 \cdot (\log n + c) \cdot p)$,
2. Cost of fix function invocation: $(\lceil \frac{m}{MAX\_INT} \rceil \cdot c_4 \cdot c) + (c \cdot c_5)$, that is in $\mathcal{O}(n \cdot c + c)$, where MAX_INT it is the biggest value representable by integers; when m exceeds MAX_INT, multiple integers are used to express the encoding,
3. Cost of fitness function invocation: its complexity is equal to $m \cdot c_6$, that is in $\mathcal{O}(n)$,
4. Cost of computing generations $[2 \dots it]$: $(it-1) \cdot (p-r) \cdot m \cdot (c_2 + c_3) + \mathsf{fix}() + \mathsf{fitness}()$, that is in $\mathcal{O}(it \cdot p \cdot n^2 \cdot c)$.

Total time complexity for the GA-based heuristic is thus in $\mathcal{O}((n^2 \cdot (\log n + c) \cdot p) + (it \cdot p \cdot n^2 \cdot c))$, that is equal to $\mathcal{O}(n^2 \cdot p \cdot ((\log n + c) + it \cdot c))$. This proves the proposition.

□

In Proposition C.5 the temporal complexity of the CycleBreaker variant shown in Algorithm 13 is analyzed.

Proposition C.5. Given a digraph G, the time complexity of CycleBreaker algorithm variant described in Algorithm 13 is in $\mathcal{O}(n^2 \cdot (c+1))$ (resp. $\mathcal{O}(n^2 \cdot p \cdot ((\log n + c) + it \cdot c))$), where $n = |A| + |V|$, when solver function is implemented using the greedy heuristic (resp. the GA-based heuristic). When the *ASP* solver is employed, the complexity is dominated by that of *MAP-WFES*/*CMAP-WFES*, that are *NP*-hard problems.

Proof. First we derive an estimation of the average size j of each SCC, given the total number i of SCCs in the graph G: $1 \leq i = |globalSCCs| \leq |V|$, $j = |V|/i = avg(|scc|)$, where $scc \in globalSCCs$.

line 2: createDigraph $\in \mathcal{O}(n)$, Proposition C.1,

lines 3–4: SCC $\in \mathcal{O}(n)$ [Tar72],

lines 8–16: the loop calls i times the following functions:

- project and membership, that can be precomputed and accessed in constant time, c_1,
- extractMappings, that can be precomputed and accessed in constant time, c_2,
- johnson, whose time complexity is in $\mathcal{O}(n \cdot (c+1))$, where c is the number of distinct cycles in G,
- the complexity of solver function depends on its underlying implementation; when solver is implemented using:
  1. an *ASP* solver, the complexity is dominated by that of *MAP-WFES* and *CMAP-WFES*, that are *NP*-hard problems,
  2. the greedy heuristic, the total complexity is equal to $2n + i \cdot (c_1 + c_2 + (n \cdot (c + 1)) + m \cdot (\log m + c))$, that is in $\mathcal{O}(n^2 \cdot (c + 1))$, given that $i \leq |V| \leq n$ and $n > m$,
  3. the GA-based heuristic, the total temporal complexity is $2n + i \cdot (c_1 + c_2 + (n \cdot (c + 1)) + GA)$, that is in $\mathcal{O}(n^2 \cdot c + GA)$. Total time complexity is clearly dominated by the complexity of the GA-based heuristic, and it is therefore in $\mathcal{O}(n^2 \cdot p \cdot ((\log n + c) + it \cdot c))$.

This proves the proposition.

□

It is of interest to remark that, despite the *CMAP-WFES* and *MAP-WFES* are *NP*-hard problems, the practical complexity is sensibly lowered by using an highly optimized *ASP* solver, as experimentally verified in Section 5.5.3.

In Proposition C.6, the temporal complexity of filterMultipleOccurrences function, used in Algorithm 14, is investigated.

Proposition C.6. Given an alignment $\mathcal{M}$, its associated graph representation $G = (V, A)$, and given also a subset $\mathcal{M}_s$ of $\mathcal{M}$, denoted as the set of problematic mappings, the time complexity of filterMultipleOccurrences function (defined in Algorithm 15) is in $\mathcal{O}(m \log m)$, where $m = |\mathcal{M}_s|$.

Proof.

1. For each element of $\mathcal{M}_s$ (lines 4– 11) we compute its hash-value and we add the computed pair to (the head/tail of) list L, that requires constant time c_1, totalling $c_1 \cdot n$,
2. sorting list L can be done in $\mathcal{O}(n \log n)$ (*e.g.*, using *Merge Sort* algorithm),
3. for each element of L (lines 14–25), a constant (c_2) number of operations is performed (element removal can be performed in constant time on hash-sets), totalling $c_2 \cdot m$.

The total complexity is therefore equal to $2 \cdot (c_1 \cdot m + m \log m + c_2 \cdot n)$, that is in $\mathcal{O}(m \log m)$. This concludes the proof.

□

From the results of Proposition C.1 and Proposition C.6, it is evident that the CycleBreaker variant shown in Algorithm 14 is dominated by the (higher) complexity of the employed solver.

Bibliography

[ABG05] Nikolaus Augsten, Michael Böhlen, and Johann Gamper. Approximate Matching of Hierarchical Data Using pq-Grams. In *International Conference on Very Large Data Bases (VLDB)*, pages 301–312, 2005.

[ABG10] Nikolaus Augsten, Michael Böhlen, and Johann Gamper. The pq-gram Distance between Ordered Labeled Trees. *ACM Transactions on Database Systems (TODS)*, 35(1):4, 2010.

[ABJ89] Rakesh Agrawal, Alexander Borgida, and Hosagrahar V. Jagadish. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. In *ACM SIGMOD Conference on Management of Data*, pages 253–262, 1989.

[ABM08] Yuan An, Alex Borgida, and John Mylopoulos. Discovering and Maintaining Semantic Mappings between XML Schemas and Ontologies. *Journal of Computing Science Engineering*, 2(1):44–73, 2008.

[ABS14] Joshua Amavi, Béatrice Bouchou, and Agata Savary. On Correcting XML Documents with Respect to a Schema. *The Computer Journal*, 57(5):639–674, 2014.

[ACHFR14] Joshua Amavi, Jacques Chabin, Mirian Halfeld Ferrari, and Pierre Réty. A ToolBox for Conservative XML Schema Evolution and Document Adaptation. In *Database and Expert Systems Applications (DEXA)*, pages 299–307, 2014.

[AEE+12] José-Luis Aguirre, Kai Eckert, Jérôme Euzenat, Alfio Ferrara, Willem Robert van Hage, Laura Hollink, Christian Meilicke, Andriy Nikolov, Dominique Ritze, François Scharffe, Pavel Shvaiko, Ondrej Sváb-Zamazal, Cássia Trojahn dos Santos, Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Benjamin Zapilko. Results of the Ontology Alignment Evaluation Initiative 2012. In *Ontology Matching Workshop (OM)*, 2012.

[AR13] Patrick Arnold and Erhard Rahm. Semantic Enrichment of Ontology Mappings: A Linguistic-Based Approach. In *Advances in Databases and Information System - East European Conference (ADBIS)*, pages 42–55, 2013.

[BBC+11] Mohamed Amine Baazizi, Nicole Bidoit, Dario Colazzo, Noor Malla, and Marina Sahakyan. Projection for XML update optimization. In *International Conference on Extending Database Technology (EDBT)*, pages 307–318, 2011.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL Envelope. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 364–369, 2005.

[BBL08] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL Envelope Further. In Kendall Clark and Peter F. Patel-Schneider, editors, *OWLED Workshop on OWL: Experiences and Directions*, 2008.

[BC09] Michael Benedikt and James Cheney. Semantics, Types and Effects for XML Updates. In *Database Programming Languages (DBPL)*, pages 1–17, 2009.

[BCCN13] Véronique Benzaken, Giuseppe Castagna, Dario Colazzo, and Kim Nguyen. Optimizing XML Querying Using Type-based Document Projection. *ACM Transactions on Database Systems (TODS)*, 38(1):4, 2013.

[BCG+11] Iovka Boneva, Anne-Cécile Caron, Benoît Groz, Yves Roos, Sophie Tison, and Sławek Staworko. View Update Translation for XML. In *International Conference on Database Theory (ICDT)*, pages 42–53. ACM, 2011.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BCM+13] Nicole Bidoit, Dario Colazzo, Noor Malla, Federico Ulliana, Maurizio Nolé, and Carlo Sartiani. Processing XML Queries and Updates on Map/Reduce Clusters. In *International Conference on Extending Database Technology (EDBT/ICDT)*, pages 745–748, 2013.

[BD13] Mouhamadou Ba and Gayo Diallo. Large-scale Biomedical Ontology Matching with ServOMap. *IRBM*, 2013.

[BH12] Elena Beisswanger and Udo Hahn. Towards Valid and Reusable Reference Alignments: Ten Basic Quality Checks for Ontology Alignments and their Application to Three Different Reference Data Sets. *Journal of Biomedical Semantics*, 3(Suppl 1):S4, 2012.

[BKMW01] Anne Brüggemann-Klein, Makoto Murata, and Derick Wood. Regular Tree and Regular Hedge Languages over Unranked Alphabets: Version 1. Technical Report HKUST-TCSC-2001-05, Hong Kong University of Science and Technology, Hong Kong SAR, China, May 2001.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[BM09] Dilek Basci and Sanjay Misra. Measuring and evaluating a design complexity metric for xml schema documents. *Journal of Information Science and Engineering*, 25(5):1405–1425, 2009.

[BNVdB04] Geert Jan Bex, Frank Neven, and Jan Van den Bussche. Dtds versus xml schema: a practical study. In *International Workshop on the Web and Databases (WebDB)*, pages 79–84. ACM, 2004.

[Bod04] Olivier Bodenreider. The Unified Medical Language System (UMLS): integrating biomedical terminology. *Nucleic Acids Research*, 32:267–270, 2004.

[BS03] Alexander Borgida and Luciano Serafini. Distributed Description Logics: Assimilating Information from Peer Sources. *Journal on Data Semantics*, 1:153–184, 2003.

[CAH02] Gregory Cobena, Talel Abdessalem, and Yassine Hinnach. A Comparative Study for XML Change Detection. In *Bases de Donnes Avances (BDA)*, 2002.

[CAM02] Gregory Cobena, Serge Abiteboul, and Amélie Marian. Detecting Changes in XML Documents. In *International Conference on Data Engineering (ICDE)*, pages 41–52, 2002.

[Cav13] Federico Cavalieri. *Updates on XML Documents and Schemas*. PhD thesis, University of Genova, 2013. `ftp://ftp.disi.unige.it/person/SolimandoA/CavalieriPhDThesis.pdf`.

[CDE+13] Bernardo Cuenca Grau, Zlatan Dragisic, Kai Eckert, Jérôme Euzenat, Alfio Ferrara, Roger Granada, Valentina Ivanova, Ernesto Jiménez-Ruiz, Andreas Oskar Kempf, Patrick Lambrix, Andriy Nikolov, Heiko Paulheim, Dominique Ritze, François Scharffe, Pavel Shvaiko, Cássia Trojahn dos Santos, and Ondrej Zamazal. Results of the Ontology Alignment Evaluation Initiative 2013. In *Ontology Matching Workshop (OM)*, pages 61–100, 2013.

[CDG+07] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree Automata Techniques and Applications. `http://www.grappa.univ-lille3.fr/tata`, 2007. October, 12th.

[CDL+09] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. Ontologies and Databases: The DL-Lite Approach. In *Reasoning Web. Semantic*

Technologies for Information Systems, International Summer School Lectures, pages 255–356, 2009.

[CGM97] Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful Change Detection in Structured Data. In *ACM SIGMOD Record*, volume 26, pages 26–37. ACM, 1997.

[CGM11a] Federico Cavalieri, Giovanna Guerrini, and Marco Mesiti. Dynamic Reasoning on XML Updates. In *International Conference on Extending Database Technology (EDBT)*, pages 165–176, 2011.

[CGM11b] Federico Cavalieri, Giovanna Guerrini, and Marco Mesiti. Updating XML Schemas and Associated Documents Through eXup. In *International Conference on Data Engineering (ICDE)*, pages 1320–1323, 2011.

[CGM14] Federico Cavalieri, Giovanna Guerrini, and Marco Mesiti. XSPath: Navigation on XML Schemas Made Easy. *IEEE Transactions on Knowledge Data Engineering*, 26(2):485–499, 2014.

[Cha99] Sudarshan S. Chawathe. Comparing Hierarchical Data in External Memory. In *International Conference on Very Large Data Bases*, pages 90–101, 1999.

[CHKS07] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Just the Right Amount: Extracting Modules from Ontologies. In *International Conference on World Wide Web (WWW)*, pages 717–726. ACM, 2007.

[CHKS08] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence Research (JAIR)*, 31:273–318, 2008.

[CM01] James Clark and Makoto Murata. RELAX NG Specification. `http://www.relaxng.org/spec-20011203.html`, 2001.

[CO14] Sara Cohen and Nerya Or. A General Algorithm for Subtree Similarity-Search. In *International Conference on Data Engineering (ICDE)*, pages 928–939. IEEE, 2014.

[CPST03] Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, and Sotirios Tourtounis. On Labeling Schemes for the Semantic Web. In *International World Wide Web Conference (WWW)*, pages 544–555, 2003.

[CRGMW96] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change Detection in Hierarchically Structured Information. In *AMC SIGMOD Record*, pages 493–504, 1996.

[CSG13a] Federico Cavalieri, Alessandro Solimando, and Giovanna Guerrini. Synthetising changes in xml documents as puls. *Proceedings of the VLDB Endowment*, 6(13):1630–1641, 2013.

[CSG13b] Federico Cavalieri, Alessandro Solimando, and Giovanna Guerrini. Synthetising Changes in XML Documents as PULs Extended Experiments. Technical report, 2013. `ftp://ftp.disi.unige.it/person/SolimandoA/puldiffExtExp.pdf`.

[DDRP+14] Duy Dinh, Julio Cesar Dos Reis, Cédric Pruskia, Marcos Da Silveira, and Chantal Reynaud-Delaître. Identifying Relevant Concept Attributes to Support Mapping Maintenance under Ontology Evolution. *Jorunal of Web Semantics: Science, Services and Agents on the World Wide Web*, 2014.

[DEE+14] Zlatan Dragisic, Kai Eckert, Jérôme Euzenat, Daniel Faria, Alfio Ferrara, Roger Granada, Valentina Ivanova, Ernesto Jimenez-Ruiz, Andreas Kempf, Patrick Lambrix, Stefano Montanelli, Heiko Paulheim, Dominique Ritze, Pavel Shvaiko, Alessandro Solimando, Cassia Trojahn, Ondrej Zamazal, and Bernardo Cuenca Grau. Results of the Ontology Alignment Evaluation Initiative 2014. In *Ontology Matching Workshop (OM)*, pages 61–104, 2014.

[DEST11] Jérôme David, Jérôme Euzenat, François Scharffe, and Cássia Trojahn. The Alignment API 4.0. *Semantic Web Journal*, 2(1):3–10, 2011.

[DG84] William F. Dowling and Jean H. Gallier. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[DMRW09] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An Optimal Decomposition Algorithm for Tree Edit Distance. *ACM Transactions on Algorithms*, 6(1), 2009.

[Don09] Don Chamberlin and Michael Dyck and Dana Florescu and Jim Melton and Jonathan Robie and Jérôme Siméon. XQuery Update Facility 1.0. `http://www.w3.org/TR/2009/CR-xquery-update-10-20090609/`, 2009.

[DRDP+13] Julio Cesar Dos Reis, Duy Dinh, Cédric Pruski, Marcos Da Silveira, and Chantal Reynaud-Delaître. Mapping Adaptation Actions for the Automatic Reconciliation of Dynamic Ontologies. In *International Conference on Conference on Information and Knowledge Management (CIKM)*, pages 599–608. ACM, 2013.

[DRPDSRD12] Julio Cesar Dos Reis, Cédric Pruski, Marcos Da Silveira, and Chantal Reynaud-Delaître. Analyzing and Supporting the Mapping Maintenance Problem in Biomedical Knowledge Organization Systems. In *Workshop on Semantic Interoperability in Medical Informatics (SIMI)*, 2012.

[ECM] ECMA International. The JSON Data Interchange Format. `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`.

[EMS+11] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn. Ontology Alignment Evaluation Initiative: Six Years of Experience. *Journal on Data Semantics*, 15:158–192, 2011.

[ENSS98] Guy Even, J Seffi Naor, Baruch Schieber, and Madhu Sudan. Approximating Minimum Feedback Sets and Multicuts in Directed Graphs. *Algorithmica*, 20(2):151–174, 1998.

[ES10] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer, 2010.

[Euz07] Jérôme Euzenat. Semantic Precision and Recall for Ontology Alignment Evaluation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 348–353, 2007.

[Fab13] Wolfgang Faber. Answer Set Programming. In *Reasoning Web*, pages 162–193, 2013.

[Fon01] Robin La Fontaine. A Delta Format for XML: Identifying Changes in XML Files and Representing the Changes in XML. In *XML Europe*, 2001.

[FPS+13] Daniel Faria, Catia Pesquita, Emanuel Santos, Matteo Palmonari, Isabel F. Cruz, and Francisco M. Couto. The AgreementMakerLight Ontology Matching System. In *OTM Conferences*, pages 527–541, 2013.

[FR12] Sébastien Ferré and Sebastian Rudolph. Advocatus Diaboli - Exploratory Enrichment of Ontologies with Negative Constraints. In *International Conference on Knowledge Engineering (EKAW)*, pages 42–56, 2012.

[FV11] Daniel Fleischhacker and Johanna Völker. Inductive Learning of Disjointness Axioms. In *OTM Conferences*, pages 680–697, 2011.

[GDRH+13] Anika Groß, Julio Cesar Dos Reis, Michael Hartung, Cédric Pruski, and Erhard Rahm. Semi-automatic Adaptation of Mappings between Life Science Ontologies. In *Data Integration in the Life Sciences (DILS)*, pages 90–104. Springer, 2013.

[GH88] David E. Goldberg and John H. Holland. Genetic Algorithms and Machine Learning. *Machine Learning Journal*, 3(2):95–99, 1988.

[GHJR+15] Martin Giese, Peter Haase, Ernesto Jiménez-Ruiz, Davide Lanti, Özgür L. Özçep, Martin Rezk, Riccardo Rosati, Ahmet Soylu, Guillermo Vega-Gorgojo, Arild Waaler, and Guohui Xiao. Optique: Zooming In on Big Data Access. 2015. IEEE Computer, Accepted.

[GHWKS10] Bernardo Cuenca Grau, Christian Halaschek-Wiener, Yevgeny Kazakov, and Boontawee Suntisrivaraporn. Incremental Classification of Description Logics Ontologies. *Journal of Automated Reasoning*, 44(4):337–369, 2010.

[GKK+11] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The Potsdam Answer Set Solving Collection. *Ai Communications*, 24(2):107–124, 2011.

[GLB13] Philippe Galinier, Eunice Lemamou, and Mohamed Wassim Bouzidi. Applying Local Search to the Feedback Vertex Set Problem. *Journal of Heuristics*, pages 1–22, 2013.

[GLQ11] Pierre Genevès, Nabil Layaïda, and Vincent Quint. Impact of XML Schema Evolution. *ACM Transactions on Internet Technology (TOIT)*, 11(1):4–35, 2011.

[GMS07] Giovanna Guerrini, Marco Mesiti, and Matteo Sorrenti. XML Schema Evolution: Incremental Validation and Efficient Document Adaptation. In *XSym*, pages 92–106, 2007.

[GPS12] Rafael S Gonçalves, Bijan Parsia, and Ulrike Sattler. Concept-based semantic difference in expressive description logics. In *International Semantic Web Conference (ISWC)*, pages 99–115. Springer, 2012.

[Gru93] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition Journal*, 5(2):199–220, June 1993.

[HB11] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web Journal*, 2(1):11–21, 2011.

[HGR10] Michael Hartung, Anika Groß, and Erhard Rahm. Rule-based Generation of Diff Evolution Mappings Between Ontology Versions. *arXiv preprint arXiv:1010.0122*, 2010.

[HGR12] Michael Hartung, Anika Gross, and Ehrard Rahm. COnto–Diff: Generation of Complex Evolution Mappings for Life Science Ontologies. *Journal of Biomedical Informatics*, 2012.

[HKM+12] Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert E Tarjan. Incremental Cycle Detection, Topological Ordering, and Strong Component Maintenance. *Transactions on Algorithms (TALG)*, 8(1):3, 2012.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more Irresistible SROIQ. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 57–67, 2006.

[HM02] Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell, 2nd Edition*. OReilly Publishing, 2002.

[Hor11] Matthew Horridge. *Justification Based Explanation in Ontologies*. PhD thesis, University of Manchester, 2011.

[HPS08] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and Precise Justifications in OWL. *International Semantic Web Conference (ISWC)*, pages 323–338, 2008.

[HPS10] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification Oriented Proofs in OWL. *International Semantic Web Conference (ISWC)*, pages 354–369, 2010.

[IET] IETF as M. Mealling and R. Denenberg. Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations. `http://tools.ietf.org/html/rfc3305`.

[IL13] Valentina Ivanova and Patrick Lambrix. A Unified Approach for Aligning Taxonomies and Debugging Taxonomies and their Alignments. In *European Semantic Web Conference (ESWC)*, pages 1–15. Springer, 2013.

[Jam99] James Clark and Steve DeRose. XML Path Language (XPath) V1.0. `http://www.w3.org/TR/xpath/`, 1999.

[JMSK09] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology Matching With Semantic Verification. *Journal of Web Semantic*, 7(3):235–251, 2009.

[Joh75] Donald B. Johnson. Finding all the Elementary Circuits of a Directed Graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[JR10] Florent Jacquemard and Michael Rusinowitch. Rewrite-based Verification of XML Updates. In *International Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 119–130. ACM, 2010.

[JRC11] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. LogMap: Logic-based and Scalable Ontology Matching. In *International Semantic Web Conference (ISWC)*, pages 273–288, 2011.

[JRCH12] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Ian Horrocks. On the Feasibility of Using OWL 2 DL Reasoners for Ontology Matching Problems. In *OWL Reasoner Evaluation Workshop (ORE)*, 2012.

[JRCHB09] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Ian Horrocks, and Rafael Berlanga. Ontology integration using mappings: Towards getting the right logical consequences. In *European Semantic Web Conference (ESWC)*, pages 173–187, 2009.

[JRCHB11a] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Ian Horrocks, and Rafael Berlanga. Logic-based Assessment of the Compatibility of UMLS Ontology Sources. *Journal of Biomedical Semantics*, 2(Suppl 1):S2, 2011.

[JRCHB11b] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Ian Horrocks, and Rafael Berlanga. Supporting Concurrent Ontology Development: Framework, Algorithms and Tool. *Data & Knowledge Engineering*, 70(1):146–164, 2011.

[JRCZH12] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Yujiao Zhou, and Ian Horrocks. Large-scale Interactive Ontology Matching: Algorithms and Implementation. In *European Conference on Artificial Intelligence (ECAI)*, pages 444–449, 2012.

[JRMCH13] Ernesto Jiménez-Ruiz, Christian Meilicke, Bernardo Cuenca Grau, and Ian Horrocks. Evaluating Mapping Repair Systems with Large Biomedical Ontologies. In *Description Logics (DL)*, pages 246–257, 2013.

[JV92] Guy Jacobson and Kiem-Phong Vo. Heaviest Increasing/Common Subsequence Problems. In *Combinatorial Pattern Matching*, pages 52–66, 1992.

[KGH11] Ilianna Kollia, Birte Glimm, and Ian Horrocks. SPARQL query answering over OWL ontologies. In *The Semantic Web: Research and Applications*, pages 382–396. Springer, 2011.

[KJRZ+13] Evgeny Kharlamov, Ernesto Jiménez-Ruiz, Dmitriy Zheleznyakov, Dimitris Bilidas, Martin Giese, Peter Haase, Ian Horrocks, Herald Kllapi, Manolis Koubarakis, Özgür L. Özçep, Mariano Rodriguez-Muro, Riccardo Rosati, Michael Schmidt, Rudolf Schlatte, Ahmet Soylu, and Arild Waaler. Optique: Towards OBDA Systems for Industry. In *European Semantic Web Conference (ESWC) Satellite Events*, pages 125–140, 2013.

[KKS11] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. Concurrent Classification of EL Ontologies. In *International Semantic Web Conference (ISWC)*, pages 305–320, 2011.

[KPHS07] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. *International Semantic Web Conference (ISWC)*, pages 267–280, 2007.

[KPL+11] Asad Masood Khattak, Zeeshan Pervez, Khalid Latif, AM Jehad Sarkar, Sungyoung Lee, and Young-Koo Lee. Reconciliation of Ontology Mappings to Support Robust Service Interoperability. In *International Conference on Services Computing (SCC)*, pages 298–305. IEEE, 2011.

[KSH12] Markus Krötzsch, František Simančík, and Ian Horrocks. A Description Logic Primer. *CoRR*, abs/1201.4089, 2012.

[KWW08] Boris Konev, Dirk Walther, and Frank Wolter. The Logical Difference Problem for Description Logic Terminologies. In *International Joint Conference on Automated Reasoning (IJCAR)*, pages 259–274, 2008.

[KWZ08] Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. Can you Tell the Difference Between DL-Lite Ontologies? In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2008.

[LDI13] Patrick Lambrix, Zlatan Dragisic, and Valentina Ivanova. Get My Pizza Right: Repairing Missing Is-a Relations in $\mathcal{ALC}$ Ontologies. In *Semantic Technology*, pages 17–32. Springer, 2013.

[Lif08] Vladimir Lifschitz. What Is Answer Set Programming?. In *AAAI*, volume 8, pages 1594–1597, 2008.

[LKR05] Ralf Lammel, Stan Kitsis, and Dave Remy. Analysis of XML Schema Usage. In *International Conference on XML*, 2005.

[LL13] Patrick Lambrix and Qiang Liu. Debugging the Missing Is-a Structure Within Taxonomies Networked by Partial Reference Alignments. *Data Knowledge Engineering (DKE)*, 86:179–205, 2013.

[LW10] Carsten Lutz and Frank Wolter. Deciding Inseparability and Conservative Extensions in the Description Logic EL. *Journal of Symbolic Computing*, 45(2):194–228, 2010.

[LWH+14] Fei Li, Hongzhi Wang, Liang Hao, Jianzhong Li, and Hong Gao. Approximate joins for XML at label level. *Information Sciences Journal*, 282:237–249, 2014.

[LWKDI13] Patrick Lambrix, Fang Wei-Kleiner, Zlatan Dragisic, and Valentina Ivanova. Repairing Missing Is-a Structure in Ontologies is an Abductive Reasoning Problem. In *International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM)*, page 33, 2013.

[LWW07] Carsten Lutz, Dirk Walther, and Frank Wolter. Conservative Extensions in Expressive Description Logics. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 453–458, 2007.

[MAB+14] Viviana Mascardi, Davide Ancona, Matteo Barbieri, Rafael H. Bordini, and Alessandro Ricci. CooL-AgentSpeak: Endowing AgentSpeak-DL Agents with Plan Exchange and Ontology Services. *Web Intelligence and Agent Systems*, 12(1):83–107, 2014.

[MABR11] Viviana Mascardi, Davide Ancona, Rafael H. Bordini, and Alessandro Ricci. CooL-AgentSpeak: Enhancing AgentSpeak-DL Agents with Plan Exchange and Ontology Services. In *International Conference on Web Intelligence and Intelligent Agent Technology - Vol. 2*, pages 109–116, 2011.

[MBPS05] Sebastian Maneth, Alexandru Berlea, Thomas Perst, and Helmut Seidl. XML Type Checking with Macro Tree Transducers. In *International Symposium on Principles of Database Systems (PODS)*, pages 283–294, 2005.

[MCHS09] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, and Ulrike Sattler. Representing Ontologies using Description Logics, Description Graphs, and Rules. *Artificial Intelligence Journal*, 173(14):1275–1309, 2009.

[Mei11] Christian Meilicke. *Alignments Incoherency in Ontology Matching*. PhD thesis, University of Mannheim, 2011.

[MLMK05] Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technologies (TOIT)*, 5(4):660–704, 2005.

[MM85] Webb Miller and Eugene W. Myers. A File Comparison Program. *Software: Practice and Experience*, 15(11), 1985.

[MNS+11] Mark A. Musen, Natalya F. Noy, Nigam H. Shah, Patricia L. Whetzel, Christopher G. Chute, Margaret-Anne Story, Barry Smith, et al. The National Center for Biomedical Ontology. *Journal of the American Medical Informatics Association*, 19(2):–, 2011.

[MS09] Hélio Martins and Nuno Silva. A User-driven and a Semantic-based Ontology Mapping Evolution Approach. In *International Conference on Enterprise Information Systems (ICEIS)*, pages 214–221, 2009.

[MSH09] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research (JAIR)*, 36:165–228, 2009.

[MST09] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Reasoning Support for Mapping Revision. *Journal of Logical Computing*, 19(5), 2009.

[MSY04] Andrew McDowell, Chris Schmidt, and Kwok-bun Yue. Analysis and Metrics of XML Schema. In *Software Engineering Research and Practice (SERP)*, pages 538–544, 2004.

[MTP06] Irena Mlynkova, Kamil Toman, and Jaroslav Pokorný. Statistical Analysis of Real XML Data Collections. In *International Conference on Management of Data (COMAD)*, pages 20–31, 2006.

[Mur97a] Makoto Murata. DTD Transformation by Patterns and Contextual Conditions. In *Proceedings of SGML/XML*, volume 97, pages 325–332, 1997.

[Mur97b] Makoto Murata. Transformation of Documents and Schemas by Patterns and Contextual Conditions. In *Principles of Document Processing (PODP)*, pages 153–169, 1997.

[MVS08] Christian Meilicke, Johanna Völker, and Heiner Stuckenschmidt. Learning Disjointness for Debugging Mappings between Lightweight Ontologies. In *International Conference on Knowledge Engineering (EKAW)*, pages 93–108, 2008.

[Mye86] Eugene W. Myers. An O(ND) Difference Algorithm and its Variations. *Algorithmica*, 1(2):251–266, 1986.

[NB09] Victoria Nebot and Rafael Berlanga. Efficient Retrieval of Ontology Fragments Using an Interval Labeling Scheme. *Information Science Journal*, 179(24):4151–4173, 2009.

[NB12] DuyHoa Ngo and Zohra Bellahsene. YAM++ : A Multi-strategy Based Approach for Ontology Matching Task. In *International Conference on Knowledge Engineering (EKAW)*, pages 421–425, 2012.

[NB13] DuyHoa Ngo and Zohra Bellahsene. YAM++ results for OAEI 2013. In *Ontology Matching Workshop (OM)*, pages 211–218, 2013.

[NKMM12] Martin Necaský, Jakub Klímek, Jakub Malý, and Irena Mlýnková. Evolution and Change Management of XML-based Systems. *Journal of Systems and Software (JOSS)*, 85(3):683–707, 2012.

[NM+02] Natalya F. Noy, Mark A. Musen, et al. Promptdiff: A fixed-point Algorithm for Comparing Ontology Versions. In *National Conference on Artificial Intelligence*, pages 744–750. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[PA11] Mateusz Pawlik and Nikolaus Augsten. RTED: A Robust Algorithm for the Tree Edit Distance. *Proceedings of the VLDB Endowment*, 5(4):334–345, 2011.

[PFSC13] Catia Pesquita, Daniel Faria, Emanuel Santos, and Francisco M. Couto. To repair or not to repair: reconciling correctness and coherence in ontology reference alignments. In *Ontology Matching Workshop (OM)*, pages 13–24, 2013.

[Pri04] Priscilla Walmsley and David C. Fallside. XML Schema Part 0: Primer Second Edition. `http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/`, 2004.

[PT14] Terry R. Payne and Valentina Tamma. A Dialectical Approach to Selectively Reusing Ontological Correspondences. In *Knowledge Engineering and Knowledge Management (EKAW)*, pages 397–412. Springer, 2014.

[Rei87] Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence Journal*, 32(1), 1987.

[RS07] Mukund Raghavachari and Oded Shmueli. Efficient Revalidation of XML Documents. *Transactions on Knowledge Data Engineering (TKDE)*, 19(4):554–567, 2007.

[RSD+14] Julio Cesar Dos Reis, Marcos Da Silveira, Duy Dinh, Cedric Pruski, and Chantal Reynaud-Delaitre. Requirements for implementing mapping adaptation systems. In *International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 405–410. IEEE, 2014.

[Rud11] Sebastian Rudolph. Foundations of Description Logics. *Reasoning Web. Semantic Technologies for the Web of Data*, pages 76–136, 2011.

[RZ04] Liam Roditty and Uri Zwick. A Fully Dynamic Reachability Algorithm for Directed Graphs with an Almost Linear Update Time. In *International Symposium on Theory of Computing*, pages 184–191. ACM, 2004.

[RZ08] Liam Roditty and Uri Zwick. Improved Dynamic Reachability Algorithms for Directed Graphs. *SIAM Journal on Computing*, 37(5):1455–1471, 2008.

[S+01] Albrecht Schmidt et al. XMark - an XML benchmark project. http://www.xml-benchmark.org/, 2001.

[SC03] Stefan Schlobach and Ronald Cornet. Non-standard Reasoning Services for the Debugging of Description Logic Terminologies. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 355–362, 2003.

[Sch05] Stefan Schlobach. Debugging and Semantic Clarification by Pinpointing. In *European Semantic Web Conference (ESWC)*, pages 226–240. Springer, 2005.

[SDG12a] Alessandro Solimando, Giorgio Delzanno, and Giovanna Guerrini. Static Analysis of XML Document Adaptations. In *Advances in Conceptual Modeling (ER Workshops)*, pages 57–66. Springer, 2012.

[SDG12b] Alessandro Solimando, Giorgio Delzanno, and Guerrini Guerrini. Automata-based Static Analysis of XML Document Adaptation. In *International Symposium on Games, Automata, Logics and Formal Verification (GandALF)*, pages 85–98, 2012.

[SDG14] Alessandro Solimando, Giorgio Delzanno, and Giovanna Guerrini. Validating XML Document Adaptations via Hedge Automata Transformations. *Theoretical Compututer Science Journal (TCS)*, 560:251–268, 2014.

[SE12] Pavel Shvaiko and Jérôme Euzenat. Ontology Matching: State of the Art and Future Challenges. *Transactions on Knowledge and Data Engineering (TKDE)*, 2012.

[Sei90] Helmut Seidl. Deciding Equivalence of Finite Tree Automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.

[SFPC13] Emanuel Santos, Daniel Faria, Cátia Pesquita, and Francisco Couto. Ontology Alignment Repair Through Modularization and Confidence-based Heuristics. *arXiv:1307.5322 preprint*, 2013.

[SJG14] Alessandro Solimando, Ernesto Jiménez-Ruiz, and Giovanna Guerrini. A Multi-strategy Approach for Detecting and Correcting Conservativity Principle Violations in Ontology Alignments. In *International Workshop on OWL: Experiences and Directions (OWLED)*, pages 13–24, 2014.

[SJP14] Alessandro Solimando, Ernesto Jiménez-Ruiz, and Christoph Pinkel. Evaluating Ontology Alignment Systems in Query Answering Tasks. In *International Semantic Web Posters & Demonstrations Track (ISWC)*, pages 301–304, 2014.

[SJRG14] Alessandro Solimando, Ernesto Jiménez-Ruiz, and Giovanna Guerrini. Detecting and Correcting Conservativity Principle Violations in Ontology-to-Ontology Mappings. In *International Semantic Web Conference (ISWC)*, pages 1–16, 2014.

[Sol14] Alessandro Solimando. Detecting and Correcting Conservativity Principle Violations in Ontology Mappings. In *International Semantic Web Conference, Selected Doctoral Consortium Contributions (ISWC)*, pages 545–552, 2014.

[Sol15] Alessandro Solimando. Change Management in the Traditional and Semantic Web (Extended Experiments). Technical report, 2015. `ftp://ftp.disi.unige.it/person/SolimandoA/thesisExtExp.pdf`.

[SQJH08] Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. A Modularization-Based Approach to Finding All Justifications for OWL DL Entailments. In *Asian Semantic Web Conference (ASWC)*, pages 1–15, 2008.

[SSG+13] Ahmet Soylu, Martin G. Skjæveland, Martin Giese, Ian Horrocks, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, and Dmitriy Zheleznyakov. A Preliminary Approach on Ontology-Based Visual Query Formulation for Big Data. In *Research Conference on Metadata and Semantics Research (MTSR)*, volume 390 of *Communications in Computer and Information Science*, pages 201–212, 2013.

[SSZ09] Ulrike Sattler, Thomas Schneider, and Michael Zakharyaschev. Which Kind of Module Should I Extract? In *Description Logics Workshop (DL)*, 2009.

[Tar72] Robert Tarjan. Depth-first Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[Tho90] Wolfgang Thomas. Automata on Infinite Objects. *Handbook of theoretical computer science*, pages 133–191, 1990.

[Tim90] Tim Berners-Lee and Robert Cailliau. WorldWideWeb: Proposal for a HyperText Project. `http://www.w3.org/Proposal.html`, 1990.

[Tou12] Tayssir Touili. Computing Transitive Closures of Hedge Transformations. *International Journal of Critical Computer-Based Systems*, 3:132–150, 2012.

[Tsa14] Dmitry Tsarkov. Incremental and Persistent Reasoning in FaCT++. In *OWL Reasoner Evaluation Workshop (ORE)*, volume 1207, pages 16–22. CEUR, 2014.

[Vis06] Joost Visser. Structure Metrics for XML Schema. *National Conference XML: Aplicacoes e Tecnologias Associadas (XATA)*, 2006, 2006.

[vSB+05] Ondřej Šváb, Vojtěch Svátek, Petr Berka, Dušan Rak, and Petr Tomášek. OntoFarm: Towards an Experimental Collection of Parallel Ontologies. In *International Semantic Web Conference (ISWC). Poster Session*, 2005.

[VVSH07] Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning Disjointness. In *European Semantic Web Conference (ESWC)*, pages 175–189, 2007.

[W3C04] W3C. RDF Primer. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2004.

[W3C07] W3C. Document Type Definition (DTD). http://www.w3.org/TR/2004/REC-xml11-20040204/#NT-doctypedecl, 2007.

[W3C08] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/REC-xml, 2008.

[W3C09] W3C as Hitzler, Pascal and Krötzsch, Markus and Parsia, Bijan Patel-Schneider, Peter F. and Rudolph, Sebastian. OWL 2 Web Ontology Language Primer. http://www.w3.org/TR/owl2-primer/, 2009.

[W3C10a] W3C. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20, 2010.

[W3C10b] W3C. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery, 2010.

[W3C11] W3C. XQuery Update Facility 1.0. http://www.w3.org/TR/xquery-update-10/, 2011.

[W3C12] W3C. XML Schema 1.1. http://www.w3.org/TR/xmlschema11-1/,http://www.w3.org/TR/xmlschema11-2/, 2012.

[W3C13] W3C. XQuery 3.0: An XML Query Language. http://www.w3.org/TR/xquery-30/, 2013.

[W3C14] W3C. RDF 1.1 Primer. http://www.w3.org/TR/rdf-primer/, 2014.

[W3S] W3Schools. DTD Tutorial. http://www.w3schools.com/dtd/default.asp.

[WDC03] Yuan Wang, David J. DeWitt, and Jin-Yi Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *International Conference on Data Engineering (ICDE)*, pages 519–530, 2003.

[Win05] Marianne Winslett. Bruce lindsay speaks out. *SIGMOD Record*, 34(2):71, 2005. http://www.sigmod.org/sigmod/record/issues/0506/p71-column-winslet.pdf.

[WNS+11] Patricia L. Whetzel, Natalya F. Noy, Nigam H. Shah, Paul R. Alexander, Csongor Nyulas, Tania Tudorache, and Mark A. Musen. BioPortal: Enhanced Functionality via new Web Services from the National Center for Biomedical Ontology to Access and use Ontologies in Software Applications. *Nucleic Acids Research*, 39(suppl 2):W541–W545, 2011.

[WX12] Peng Wang and Baowen Xu. Debugging Ontology Mappings: A Static Approach. *Computing and Informatics*, 27(1):21–36, 2012.

[XWW+02] Haiyuan Xu, Quanyuan Wu, Huaimin Wang, Guogui Yang, and Yan Jia. KF-Diff+: Highly Efficient Change Detection Algorithm for XML Documents. In *International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, pages 1273–1286, 2002.

[ZL03] Silvano Dal Zilio and Denis Lugiez. XML Schema, Tree Logic and Sheaves Automata. In *Rewriting Techniques and Applications*, pages 246–263, 2003.

[ZMB04] Songmao Zhang, Peter Mork, and Olivier Bodenreider. Lessons Learned from Aligning two Representations of Anatomy. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2004.

[ZS89] Kaizhong Zhang and Dennis Shasha. Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM Journal on Computation*, 18(6):1245–1262, 1989.