UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Verfahren für die Reparatur von Ontologie-Mappings in den Lebenswissenschaften

Masterarbeit

Leipzig, 8. Juni 2015

vorgelegt von
Victor Christen
Studiengang Master Informatik

Betreuende Hochschullehrer:

Prof. Dr. Erhard Rahm
Fakultät für Mathematik und Informatik
Abteilung Datenbanken

Dr. Anika Groß
Fakultät für Mathematik und Informatik
Abteilung Datenbanken

Zusammenfassung

Diese Masterarbeit befasst sich mit Verfahren für die Reparatur von Ontologie- Mappings in den Lebenswissenschaften. Ontologien sind eine Art der Wissensrepräsentation, die für die Beschreibung und Strukturierung von Daten verwendet werden. Besonders in den Lebenswissenschaften werden Ontologien verwendet, um biologische Objekte zu beschreiben. Das Ontologie-Matching ist ein essentieller Prozess im Bereich der Datenintegration. Aufgrund der Bedeutung des Ontologie- Matching existieren auf diesem Forschungsgebiet eine immense Anzahl an Verfahren, die sich mit dieser Thematik befassen. Jedoch können die generierten Mappings eine Vielzahl von logischen Konflikten aufweisen. Aufgrund dieser Problematik sind Verfahren notwendig, die für ein gegebenes Ontologie- Mapping die Konflikte reduziert. Da die Ontologien potentiell sehr groß sein können, müssen die Reparaturverfahren effizient und effektiv realisiert werden. Diese Arbeit fokussiert sich auf die existierenden Reparaturverfahren und die Konzeption und Realisierung eines eigenen Verfahrens. Alle Verfahren werden bzgl. der Effektivität und Effizienz für sehr große Ontologien im Bereich der Lebenswissenschaften evaluiert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die heutige Gesellschaft ist als Wissensgesellschaft anzusehen. Aufgrund der enormen, zum Teil unstrukturierten Menge an Informationen spielt die Strukturierung und Beschreibung der Daten sowie die Repräsentation von Wissen eine zentrale Rolle für die automatische Verarbeitung von Informationen. Mithilfe der Aufbereitung von Daten eröffnet sich die Möglichkeit der gezielten und effizienteren Erschließung von Wissen. Der potentielle Mehrwert durch die Strukturierung von Wissen ist in vielen Bereichen der Gesellschaft präsent, wie z.B. in den Lebenswissenschaften, im E-Buisness, der Dokumentation von Geschäftsprozessen in Unternehmen, der elektronischen Anfrageverarbeitung, in den sozialen Medien etc. Eine Technologie, die sich in den vergangenen Jahren etabliert hat, ist das Semantic Web. Mithilfe des Semantic Web ist es möglich komplexe Anfragen zu unterstützen. Ein Schlüsselkonzept, das diese Technologie unterstützt, ist die Verwendung von Ontologien. Ontologien sind eine Art der Wissensrepräsentation, die verschiedene Ausprägungen haben können.

Ontologien lassen sich nach ihrer Ausdrucksmächtigkeit klassifizieren [1]. Die Vielzahl der verschiedenen Kategorien resultiert aus der Diskrepanz zwischen Anwendungsfreundlichkeit und präziser Beschreibung der Realwelt. Die trivialste Form von Ontologien sind Kataloge, die eine Menge von Termen umfassen, die aber inhaltlich nicht beschrieben sind und keine Beziehungen zueinander aufweisen. Eine semantische Beschreibung der Terme bieten Glossare und Thesauri. Jeder Term eines Glossars ist inhaltlich mit natürlich sprachlichen Mitteln beschrieben. Jedoch ist die Art der Beschreibung nicht geeignet für die automatisierte Verarbeitung durch computergesteuerte Agenten. Thesauri beinhalten Relationen zwischen den Termen, wie z.B. Synonyme, Oberbegriffe etc. Jedoch sind diese Relationen nicht formal und somit ist die automatische Verarbeitung problematisch. Mithilfe der is_a Relation können Terme in einer Hierarchie strukturiert werden. Dabei wird zwischen informalen und formalen is_a Relationen unterschieden. Bei einer informalen is_a Relation wird nicht garantiert, dass eine Instanz eines spezifischen Konzepts ebenfalls eine Instanz eines Superkonzepts ist. Im Gegensatz dazu, sind formale is_a Beziehungen als strikt anzusehen, dass heißt eine Instanz eines spezifischen Konzepts ist ebenfalls eine Instanz des generalisierenden Konzepts. In dieser Arbeit werden ausschließlich Ontologien betrachtet, die mindestens formale is_a Relationen beinhalten. Nach Gruber [2] ist eine Ontologie eine explizite, formale Spezifikation einer gemeinsamen Konzeptualisierung. Somit ist es möglich die Instanzen einer beliebigen Domäne semantisch zu beschreiben und zu strukturieren. Jedes Konzept wird explizit durch eine Menge von Eigenschaften charakterisiert und kann induktiv durch andere Konzepte und Relationen mithilfe der Web Ontology Language (OWL) konstruiert werden. Relationen repräsentieren in einer Ontologie semantische Beziehungen zwischen je zwei Konzepten. Durch die Relationen und Konzepte der Ontologie wird ein axiomatisches System geschaffen, welches auf der Beschreibungslogik basiert. Aufgrund des wohldefinierten Formalismus ist eine Ontologie maschinell lesbar, so dass eine automatische Verarbeitung möglich ist. Mithilfe des axiomatischen Aufbaus sind Schlussfolgerungsverfahren, auch Reasoning genannt, anwendbar. Des Weiteren kann eine Ontologie als gemeinsamer Konsens in der Domäne angesehen werden, so dass verschiedene Akteure oder Forschungsgruppen ihr Wissen beschreiben können und untereinander austauschen können.

Beispielhaft für die Anwendung von Ontologien sind die Lebenswissenschaften. Dieser Bereich generiert ein hohes Datenvolumen, das biologische Objekte, Experimentdaten, klinische Dokumentationen, etc. beinhaltet. So wird z.B. die *Gene Ontology* verwendet, um Gene bzgl. ihrer Funktionalität zu annotieren. Dadurch können Wissensbasen erzeugt werden, die komplexe Anfragen bzgl. der biologischen Daten unterstützen.

Eine Ontologie umfasst im Allgemeinen nicht das gesamte Wissen einer Domäne. Um weitere Aspekte zu betrachten, ist es notwendig mehrere Ontologien zu verwenden. Des Weiteren existieren in einer Domäne diverse Datenquellen, deren Daten mithilfe von verschiedenen Ontologien beschrieben werden. Für die Integration von mehreren Datenquellen kann vorab eine Ontologie erstellt werden, die mehrere Ontologien semantisch abdeckt. Das Integrieren von Ontologien wird durch Merging- Verfahren wie z.B. in [3] ermöglicht. Für das Merging und andere Datenintegrationsszenarien ist es notwendig eine Menge von semantischen Korrespondenzen zwischen den Konzepten der Ontologien zu bestimmen. Die Menge der Korrespondenzen wird als Ontologie- Mapping oder Ontologie- Alignment bezeichnet. Da Ontologien sehr groß sein können, ist eine manuelle Identifikation nicht durchführbar. Deshalb werden (semi)- automatische Verfahren für die Bestimmung eines Mappings verwendet. Der Prozess, den diese Verfahren realisieren, wird als Ontologie- Matching bezeichnet. Die konkrete Aufgabe des Ontologie-Matching für zwei gegebene Ontologien O_1 und O_2 ist die Identifikation aller Korrespondenzen zwischen den Konzepten von O_1 und O_2, die semantisch korrespondieren. Es existiert eine Vielzahl von Verfahren (siehe Übersicht [4]), die sich mit dieser Thematik befassen.

Die Qualität eines Mappings wird durch die Korrektheit, die Vollständigkeit und die logische Konsistenz charakterisiert. Ein Mapping ist *korrekt*, wenn alle identifizierten Korrespondenzen richtig sind. Ein Maß für die Korrektheit ist die *Precision*. Ein Mapping ist *vollständig*, wenn alle real existierenden Korrespondenzen identifiziert wurden. Des Weiteren ist es essentiell, dass das generierte Mapping *logisch konsistent* ist. Unter der Annahme, dass die Quellontologien logisch konsistent sind, führt ein fehlerbehaftetes Alignment zu einer logisch konfliktbehafteten Wissensbasis. Jedoch existieren in der Praxis Ontologien, die nicht logisch korrekt sind, da der Fokus bei der Modellierung auf die Benutzerfreundlichkeit gerichtet ist. Die Art der logischen Inkorrektheit wird in Inkonsistenz und Inkohärenz differenziert. Jegliche Reasoning- Verfahren, die auf einem Widerspruchsbeweis basieren, sind bei inkonsistenten Ontologien nicht anwendbar. Eine inkohärente Ontologie beinhaltet Konzepte, die unerfüllbar sind, dass bedeutet das diese Konzepte nicht für die Annotation verwendet werden können.
Darüber hinaus sollten die Verfahren effizient sein und auch für große Ontologien skalieren. Da eine Ontologie viele Entitäten enthält, führen naive Ansätze zu hohen Laufzeiten. Daher werden spezielle Techniken verwendet, die die Anzahl der Vergleiche reduzieren oder den Match-Prozess parallelisieren.
Um der Problematik bzgl. der logischen Inkonsistenz eines Mappings entgegen zu wirken, ist es notwendig Verfahren zu entwickeln, die ein bestehendes Alignment $\mathcal{M}$ für zwei Ontologien O_1 und O_2 korrigieren. Im Gegensatz zu den Merging- Verfahren, verwenden die Reparaturverfahren ein vereinfachtes Integrationsmodell der Wissensbasen der Form $O_1 \cup O_2 \cup \mathcal{M}$, wobei

die Konzepte der zwei Ontologien durch die Korrespondenzen des Mappings $\mathcal{M}$ miteinander verknüpft werden.

1.2 Ziele der Arbeit

Das Ziel dieser Arbeit ist die Konzeption und Implementierung eines generischen Reparaturverfahrens, welches ein bestehendes Alignment korrigiert. Das Verfahren soll als Post- Processing Schritt in die *Match* Komponente des Gomma Systems integriert werden. Das korrigierte Alignment und die zwei alignierten Ontologien sollen in der Vereinigung eine geringe Anzahl an logischen Konflikten aufweisen. Des Weiteren muss das Verfahren effizient und skalierbar sein, um als Post- Processing Schritt für bestehende Matching- Verfahren zu fungieren. Um die Effizienz des Verfahrens zu ermitteln, wird das Verfahren mit bestehenden Reparaturverfahren verglichen. Für den Vergleich werden die Ontologien des Large BioMed Track von der *Ontology Aligment Evaluation Initiative(OAEI)* verwendet und die Alignments, die von den Match-Systemen YAM++, ServOMap und der *Match* Komponente des GOMMA Systems berechnet wurden. Die Arbeit betrachtet dabei folgende Aspekte:

- Es sollen die existierenden Verfahren für die Reparatur von Ontologie- Alignments analysiert werden.
- Der Fokus der Arbeit ist die Konzeption und Realisierung eines effizienten Algorithmus für die Identifikation logisch inkorrekter Korrespondenzen.
- Das entwickelte Verfahren und die anderen Verfahren sollen ausführlich evaluiert werden, um die Effektivität und Effizienz darzulegen. Diesbezüglich wird vorab eine gute Konfiguration für das implementierte Verfahren ermittelt, die das bestmögliche Ergebnis erzielt.
- Das implementierte Verfahren soll als Modul in das bestehende GOMMA System integriert werden.

1.3 Aufbau der Arbeit

Das zweite Kapitel befasst sich mit Ontologien und dem Konzept der Beschreibungslogik, die die Basis von komplexen Ontologien darstellt. Des Weiteren werden die Grundlagen erläutert, die für die Realisierung des Verfahrens und die Evaluierung notwendig sind. Zu diesen Grundlagen gehören die Aspekte des Ontologie- Matching und Reasoning- Verfahren. Das dritte Kapitel befasst sich mit Mapping- Reparaturverfahren. Diesbezüglich wird erläutert, welche Ziele ein Reparaturverfahren hat und welche Probleme auftreten können. Des Weiteren werden die drei existierenden Verfahren vorgestellt. Anschließend wird die Konzeption und die Realisierung eines eigenen graphbasierten Verfahrens erläutert. Das fünfte Kapitel beinhaltet eine vergleichende Evaluierung der Verfahren. Im letzten Kapitel werden die betrachteten Aspekte und Ergebnisse der Arbeit resümiert. Des Weiteren wird ein Ausblick vorgestellt, der sich mit zukünftigen Verbesserungen und Optimierung auseinander setzt.

Kapitel 2

Grundlagen

2.1 Ontologien und Beschreibungslogik

Der Begriff der Ontologie stammt ursprünglich aus der griechischen Philosophie und bezeichnet die Lehre des Seins. In der Informatik werden Ontologien als Wissensrepräsentation verwendet. Für den Austausch von Wissen bzgl. einer Domäne ist ein einheitlich beschreibendes Format notwendig [2], um die Heterogenität der verschiedenen Wissensrepräsentations(WR)- Systeme zu überwinden. Durch die Verwendung von Ontologien existiert ein beschreibendes Format, das ebenfalls als Vokabular verwendet werden kann, um komplexes Wissen zu beschreiben. Für die Beschreibung von Ontologien existieren verschiedene Beschreibungssprachen. Komplexe Formate sind RDFS(Ressource Description Framework Schema) und OWL (Web Ontology Lanaguage), die auf XML basieren. Da diese Formate sehr komplex sind und in einigen Anwendungsfällen, wie z.B. in den Lebenswissenschaften, diese Komplexität nicht notwendig ist, wird das obo- Format verwendet. Dieses Format stellt eine textuelle Spezifikation der Ontologie dar, die nicht die komplette Ausdrucksmächtigkeit von OWL umfasst. Die Vorteile dieses Formats sind die Kompaktheit und die einfache Verständlichkeit, so dass es in der Domäne der Lebenswissenschaften weit verbreitet ist. Aufgrund der dedizierten Menge von Formaten ist eine maschinelle Verarbeitung mittels entsprechender Parser realisierbar. Komplexe Ontologien sind eine Art der Wissensrepräsentation, die der Beschreibungslogik entsprechen. Generell werden Wissensbasen in einem größeren Kontext innerhalb eines WR- Systems verwendet, welches die Möglichkeit bietet die Wissensbasis zu realisieren sowie zu manipulieren und implizites Wissen zu ermitteln. Des Weiteren umfasst jedes WR- System eine Modellierungssprache, die eine Menge von Konstrukten umfasst, um Ontologien zu definieren. Der wohldefinierte Formalismus ermöglicht die Verwendung von Reasoning- Verfahren, die in der Lage sind implizite Folgerungen zu schlussfolgern bzw. zu inferieren.

Für das weitere Verständnis werden folgende Notationen verwendet. Konzepte werden durch Großbuchstaben $A, B, C, ...$ dargestellt. Relationen werden durch Großbuchstaben $R, S, T, ...$ repräsentiert. Die Individuen oder auch Instanzen werden mit kleinen Buchstaben $a, b, c, ...$ dargestellt.

2.1.1 Grundlagen der Beschreibungslogik

Im Folgenden werden die grundlegenden Aspekte der Beschreibungslogik diskutiert. Eine detaillierte Beschreibung ist in [5] zu finden. Die Beschreibungslogik vereint zwei Aspekte. Zum einen ist es möglich, Wissen zu repräsentieren und zu beschreiben und zum anderen umfasst es einen logikbasierten Aufbau, der es ermöglicht implizites Wissen zu ermitteln. Im Wesentlichen besteht eine Wissensbasis (WB) aus einer *TBox* und einer *ABox*. Die *TBox* umfasst die gesamte Terminologie der Wissensbasis. Die Terminologie umfasst Konzepte und Relationen. Ein Konzept repräsentiert eine Menge von Instanzen, die gleiche Eigenschaften aufweisen. Analog zur Prädikatenlogik erster Stufe entsprechen Konzepte einstelligen Prädikaten. Eine Relation repräsentiert eine binäre Beziehung zwischen zwei Konzepten. Mithilfe der Relation können die Eigenschaften eines Konzepts näher spezifiziert werden. Konzepte und Relationen, die sich nicht durch andere Elemente der TBox beschreiben lassen, sind atomar. Eine essentielles Konstrukt der Beschreibunslogik ist die Subsumption bzw. auch is_a Relation. Die Subsumption

repräsentiert eine Spezifizierung eines Konzepts durch ein anderes Konzept. Die Subsumption $C_1 \sqsubseteq C_2$ bedeutet, dass das Konzept C_1 das Konzept C_2 spezifiziert. C_2 wird als Superkonzept von C_1 bezeichnet und C_1 als Subkonzept von C_2. Das Superkonzept aller Konzepte ist $\top$ und enthält alle Individuen der Domäne. $\bot$ ist das Konzept, welches keine Instanzen beinhaltet. Des Weiteren lassen sich komplexe Konzepte und Relationen induktiv definieren, basierend auf den atomaren Konzepten und Relationen sowie den Konstruktionsbausteinen der Beschreibungslogiksprache. Die Komplexität der Konzepte ist dabei abhängig von der Ausdrucksmächtigkeit der Sprache und somit von der Menge der Konstruktoren, die die Sprache zur Verfügung stellt. In der Regel werden komplexe Ontologien mithilfe von OWL spezifiziert. OWL existiert in drei Versionen: OWL Lite, OWL DL und OWL Full. Die Ontologien, die mit OWL Full beschrieben sind, sind für ein Inferenzproblem nicht entscheidbar. Im Gegensatz dazu, sind Ontologien von OWL Lite und OWL Full entscheidbar. Diese Ontologien sind jedoch in ihrer Ausdrucksmächtigkeit eingeschränkt. OWL DL entspricht der $\mathcal{SHOIN}(D)$ Beschreibungslogik und enthält die nachfolgenden Konstrukte. Jeder Buchstabe repräsentiert dabei eine Menge von Konstrukten. Zur Veranschaulichung wird eine kleine Beispielontologie verwendet.

Die Konzepte sind $Concepts = \{Student, Professor, Dozent, UniMitarbeiter, SHK, Vorlesung, Modul\}$
Die Rollen sind $Roles = \{besuchen, wirdBesucht, arbeiten, durchführen, enthalten, istTeilVon\}$

- $\mathcal{S}$
  Dieser Teil beinhaltet alle Konstrukte der $\mathcal{ALC}$ Logik sowie die Rollentransitivität. $\mathcal{ALC}$ enthält die Konjunktion, Disjunktion und Negation von Konzepten sowie die existentielle Quantifizierung und Wertrestriktion für Rollen.

  1. $UniMitarbeiter \equiv Professor \sqcup Dozent \sqcup SHK$
     Ein Universitätsmitarbeiter ist entweder ein Professor, ein Dozent oder eine SHK.
  2. $Veranstaltung \equiv Vorlesung \sqcup Seminar \sqcup Praktikum$
     Eine Veranstaltung ist entweder eine Vorlesung, ein Seminar oder ein Praktikum.
  3. $SHK \equiv UniMitarbeiter \sqcap Student$
     Eine SHK ist ein Unimitarbeiter und ein Student.
  4. $Student \sqsubseteq \neg Professor$
     Ein Student ist kein Professor.
  5. $\top \sqsubseteq \forall besuchen.Veranstaltung$
     $\exists besuchen.\top \sqsubseteq Student$
     Ein Student besucht Veranstaltungen.
  6. $\top \sqsubseteq \forall arbeiten.\top$
     $\exists arbeiten.\top \sqsubseteq UniMitarbeiter$
     Ein Uni Mitarbeiter arbeitet etwas
  7. $\top \sqsubseteq \forall durchführen.Veranstaltung$
     $\exists durchführen.\top \sqsubseteq Professor \sqcup Dozent$
     Ein Professor oder ein Dozent führt eine Veranstaltung durch.

8. $\top \sqsubseteq \forall enthalten.Veranstaltung$
   $\exists enthalten.\top \sqsubseteq Modul$
   Ein Modul enthält Veranstaltungen.
9. $\top \sqsubseteq \forall istTeilVon.Modul$
   $\exists istTeilVon.\top \sqsubseteq Modul$
   $Trans(istTeilVon)$
   Ein Modul kann Teil eines anderen Moduls sein. Wenn ein Modul C Teilmodul von B ist und B ein Teilmodul von A, dann ist C ein Teilmodul von A.

- $\mathcal{H}$ Rollenhierarchie
  durchführen $\sqsubseteq$ *arbeiten*
  Eine Veranstaltung durchführen ist eine Spezifizierung von arbeiten.

- $\mathcal{O}$ Nominale
  Nominale weisen einem Konzept ein bestimmtes Individuum zu oder ein Konzept hat eine endliche Menge an möglichen Individuen.
  $Professor \equiv \{Rahm, Fähnrich, Quasthoff\}$
  Ein Professor ist entweder Herr Rahm, Herr Fähnrich oder Herr Quasthoff.

- $\mathcal{I}$ Inverse Rollen
  Eine inverse Rolle ist die Rückrichtung einer Rolle, dass heißt der Definitionsbereich der inversen Rolle ist der Wertebereich der ursprünglichen Rolle bzw. der Wertebereich ist der Definitionsbereich.
  $wirdBesucht \equiv besuchen^{-1}$

- $\mathcal{N}$ Kardinalitätseinschränkung
  Mithilfe der Kardinalitätseinschränkung kann spezifiziert werden, wie oft ein Konzept an einer Rolle teilnehmen muss.
  Dozent $\sqsubseteq \geq 1$ *durchführen*

- D
  Dadurch sind die allgemeinen XML- Datentypen verwendbar.

Die Individuen, die in der Domäne existieren, werden zu den entsprechenden Konzepten bzw. zu den entsprechenden Rollen zugeordnet. Die Menge der Zuweisungen von Individuen zu Konzepten und Rollen werden in der *ABox* definiert. Eine ABox für die Beispielontologie ist:

$SHK(VicorChristen), Student(VictorChristen), Professor(ErhardRahm), Vorlesung(DBS\,1)$, $durchführen(ErhardRahm, DBS\,1)$

Die formale Semantik einer Ontologie wird durch eine Interpretation $\mathcal{I}$ realisiert. Eine Interpretation $\mathcal{I}$ besteht aus einer Domäne der Interpretationen $\Delta^{\mathcal{I}}$, die Individuen der Domäne enthält. Des Weiteren existiert eine Interpretationsfunktion $\cdot^{\mathcal{I}}$, die Elemente der *TBox* auf eine Teilmenge der Domäne $\Delta^{\mathcal{I}}$ abbildet. Die Abbildung 2.1 veranschaulicht den Zusammenhang zwischen den Elementen der TBox, den Elementen der Domäne und der Interpretationsfunktion. Generell werden Individuen der Ontologie auf die identischen Individuen der Domäne

abgebildet. Des Weiteren wird ein atomares Konzept auf eine Menge von Individuen abgebildet und eine atomare Relation bildet auf eine Menge von Paaren von Individuen der Domäne ab, die diese Relation erfüllen. Komplexe Konzepte und Relationen werden der Tabelle 2.1 entsprechend interpretiert. Eine Interpretation $\mathcal{I}$ erfüllt ein Axiom α ($\mathcal{I} \models \alpha$), wenn die entsprechenden Bedingungen aus der Tabelle 2.1 erfüllt sind. Wenn eine Interpretation $\mathcal{I}$ alle Axiome der Ontologie $\mathcal{O}$ erfüllt, ist $\mathcal{I}$ ein Modell für $\mathcal{O}$ ($\mathcal{I} \models \mathcal{O}$). Ein Axiom α ist eine logische Konsequenz von $\mathcal{O}$, wenn jedes Modell α erfüllt ($\mathcal{O} \models \alpha$).

Abbildung 2.1: Semantisches Modell der Beschreibungslogik. Die schwarzen Punkte repräsentieren die Individuen der Domäne und die Pfeile stellen die Interpretatio sfunktion $\cdot^{\mathcal{I}}$ für die Elemente der TBox dar.

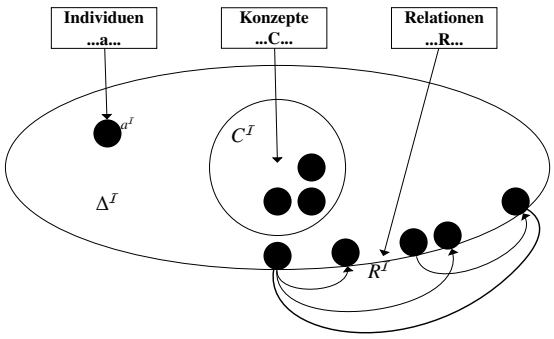


Tabelle 2.1: Interpretation komplexer Axiome der $\mathcal{SHOIN}(D)$- Beschreibungslogik

T- Box	Interpretation
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$A \sqcup B$	$A^{\mathcal{I}} \cup B^{\mathcal{I}}$
$A \sqcap B$	$A^{\mathcal{I}} \cap B^{\mathcal{I}}$
$\neg A$	$\Delta \backslash A^{\mathcal{I}}$
$\exists R.B$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b : (a,b) \in R^{\mathcal{I}} \wedge b \in B^{\mathcal{I}}\}$
$\forall R.B$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b : (a,b) \in R^{\mathcal{I}} \rightarrow b \in B^{\mathcal{I}}\}$
$Trans(R)$	$(x,y) \in R^{\mathcal{I}} \wedge (y,z) \in R^{\mathcal{I}} \rightarrow (y,z) \in R^{\mathcal{I}}$
(R^{-})	$\{(b,a) \mid (a,b) \in R^{\mathcal{I}}\}$
$(\geq n\ R.B)$	$\{a \mid \#\{b \mid (a,b) \in R \wedge b \in B\} \geq n\}$
$(\leq n\ R.B)$	$\{a \mid \#\{b \mid (a,b) \in R \wedge b \in B\} \leq n\}$

In der Praxis existieren Ontologien, die logisch nicht korrekt sind. Eine möglich Ursache sind Fehler bei der Erstellung der Ontologie. Diese Fehler können aufgrund der Komplexität der Beschreibungslogiksprache entstehen oder durch ein fehlerhaftes Alignment, wenn es sich um eine integrierte Ontologie handelt. Des Weiteren werden einige Ontologien absichtlich logisch inkorrekt modelliert, um die Benutzerfreundlichkeit zu erhöhen. So werden z.B. Konzepte in der Hierarchiestruktur von Produktkatalogen redundant gespeichert, wobei die Superkonzepte keine semantische Beziehung zueinander haben. Jedoch ist dadurch eine schnellere Suche durch den Anwender möglich. Bei der logischen Konsistenz werden zwei Arten von Konflikten betrachtet:

- *Inkonsistenz*
  Die Ontologie besitzt kein Modell, das für jedes Konzept oder jede Relation eine widerspruchsfreie Interpretation realisiert. Wenn eine Ontologie nicht konsistent ist, sind Reasoning- Verfahren, die auf einem Widerspruchsbeweis basieren, nicht anwendbar. Beim Widerspruchsbeweis wird gezeigt, dass kein Modell für die Negation des zu inferierenden Axioms und der Wissensbasis existiert. Wenn die Wissensbasis kein Modell besitzt, ist somit jedes Axiom ableitbar.

- *Inkohärenz*
  Eine Ontologie ist inkohärent, wenn mindestens ein Konzept existiert, das nicht erfüllbar ist. Ein unerfüllbares Konzept wird als leere Menge interpretiert, dass bedeutet die Wissensbasis ist ausschließlich konsistent, wenn das Konzept keine Instanzen besitzt. Durch

die unerfüllbaren Konzepte wird der Mehrwert der Ontologie reduziert, da das Hauptanwendungsgebiet einer Ontologie die Annotation von Individuen mit Konzepten ist.

2.2 Ontologie- Matching

Das Ontologie- Matching ist ein wichtiger Bestandteil für die Datenintegration, die semantische Anfrageverarbeitung, erweiternde Verfahren, wie z.B. die Änderungsbestimmung von Ontologieversionen, Analyse der Evolution von Ontologien oder die Fusionierung von Ontologien. Als Ontologie- Matching wird der semi- automatische Prozess bezeichnet, der eine Menge von Korrespondenzen zwischen den Konzepten von zwei Ontologien O_1 und O_2 ermittelt. Diese Menge der Korrespondenzen wird als Ontologie- Mapping oder als Ontologie- Alignment bezeichnet. Eine Korrespondenz repräsentiert eine Assoziation zwischen zwei Konzepten, die eine Ähnlichkeit zueinander besitzen. Die Ähnlichkeit einer Korrespondenz wird auch als *Konfidenz* bezeichnet. Ein hohe Konfidenz ist ein Beleg, dass die Korrespondenz mit hoher Wahrscheinlichkeit richtig ist. Da Ontologien sehr groß sein können, ist eine manuelle Erstellung eines Mappings nicht effizient bzw. nicht anwendbar. Deshalb ist es notwendig Verfahren bereitzustellen, die diese Aufgabe semi- automatisch lösen. Der Prozess ist semi- automatisch, da nicht garantiert werden kann, dass das erzeugte Mapping korrekt ist. Es ist weiterhin notwendig, dass die ermittelten Korrespondenzen manuell auf Korrektheit validiert werden.

Es existieren bereits zahlreiche Match- Verfahren. Da das Ontologie- Matching eine hohe Ähnlichkeit zum Schema-Matching aus dem Datenbankbereich aufweist, sind die existierenden Ontologie- Matching Verfahren durch die Taxonomie von [4] kategorisierbar. Diese Taxonomie unterteilt die Match- Verfahren in *schemabasierte*, *instanzbasierte* und *reuse orientierte* Match-Verfahren. Des Weiteren existieren Hybrid- Verfahren, die durch die Kombination von verschiedenen Match- Verfahren einen komplexen Match- Workflow realisieren.

- *schemabasiert*
  Diese Match- Verfahren operieren bei der Identifikation von Korrespondenzen ausschließlich auf den Konzepten der Terminologie der Ontologien und verwenden keine Instanzen der Konzepte. Schemabasierte Match- Verfahren unterteilen sich in element- und strukturbasierte Methoden. Bei elementbasierten Methoden werden die Eigenschaften verwendet, die ein Konzept oder eine Relation charakterisieren, wie z.B. der Name, eine Beschreibung, die Synonyme, etc. Die Ähnlichkeit solcher Eigenschaften wird durch *linguistische* Verfahren ermittelt. Des Weiteren können die *Constraints*(Einschränkungen) bzgl. der Ausprägung eines Konzepts oder einer Relation für den Vergleich verwendet werden. Bei strukturbasierten Verfahren wird die Hierarchie der Ontologien betrachtet.

- *instanzbasiert*
  Prinzipiell wird bei einem instanzbasierten Match- Verfahren für jedes Konzept eine Menge von Instanzen bereit gestellt. Ein Konzept $C_1 \in O_1$ korrespondiert zu einem Konzept $C_2 \in O_2$, wenn die Instanzmengen $C_1^{\mathcal{I}}$ und $C_2^{\mathcal{I}}$ hinreichend ähnlich sind. Die Ähnlichkeit der beiden Mengen kann mithilfe von einer Metrik wie z.B. Jaccard, Dice, etc. ermittelt werden. Diese Verfahren sind hilfreich, wenn die Entitäten auf Terminologieebe-

ne nicht hinreichend aussagekräftig sind, um ein Mapping zu bestimmen. Des Weiteren kann die Methodik verwendet werden, um ein bestehendes Alignment zu validieren, indem die Ausprägungen der Instanzen von zwei korrespondierenden Konzepten bzgl. ihrer Ähnlichkeit analysiert werden. Ähnlich wie bei *schemabasierten* Match- Verfahren wird zwischen elementbasierten und constraintbasierten Vergleichsmethoden unterschieden.

- *reuse orientiert*
  Bei dieser Variante von Match- Verfahren werden für die Generierung eines Alignments bereits berechnete Mappings verwendet. Wenn die Mappings für eine Ontologie O_1 und O_2 sowie für eine Ontologie O_2 und O_3 existieren, kann ein Mapping für O_1 und O_3 bestimmt werden. Bei der Berechnung des Mappings O_1 und O_3 werden die existierenden Mappings miteinander verknüpft (compose), so dass das neue Mapping berechnet werden kann. Eine Komposition von Mappings ist durch die Reduzierung der Vergleiche sehr effizient.

Die Qualität eines Mappings wird durch die Korrektheit, Vollständigkeit und den Grad der Inkohärenz charakterisiert. Für die Messung der Vollständigkeit und Korrektheit eines Alignments benötigt man ein Referenzalignment. Bei einem korrekten Mapping existieren keine Korrespondenzen, die falsch sind. Das Maß für die Korrektheit ist die *Precision*. Sie entspricht dem Verhältnis der Kardinalität der Menge der *richtig Positiven* (True Positive TP) und der Summe der Anzahl der richtig Positiven und der Anzahl der *falsch Positiven* (False Positive FP). Ein vollständiges Mapping enthält alle real existierenden Korrespondenzen. Ein Maß für die Vollständigkeit oder auch Abdeckung eines Mappings ist der *Recall*. Der *Recall* ist das Verhältnis der Kardinalität der Menge der *richtig Positiven*(TP) und der Summe der Anzahl der richtig Positiven und der Anzahl der falsch Negativen (False Negative FN). Da sowohl die Korrektheit als auch die Vollständigkeit wichtig sind für die Qualität eines Mappings, ist das Ziel, dass beide Maße gegen 1 konvergieren. Das harmonische Mittel der beiden Maße ist das $F - Measure$. Eine Übersicht der Gleichungen ist in Abbildung 2.2 dargestellt.

Abbildung 2.2: Berechnung von *Precision*, *Recall* und $F - Measure$

$$Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN} \qquad F - Measure = 2\frac{Precision \cdot Recall}{Precision+Recall}$$

Ein weiteres Kriterium der Qualität eines Mappings ist der Grad der Inkohärenz. Dieses Maß ist das Verhältnis der Anzahl der unerfüllbaren Konzepte und der Gesamtanzahl aller Konzepte. Diesbezüglich muss die Anzahl der unerfüllbaren Konzepte mithilfe eines Reasoners für die Wissensbasis $O_1 \cup O_2 \cup \mathcal{M}$ ermittelt werden. Im Kapitel 5 werden diese Maße für die Messung der Qualität des reparierten Mappings verwendet.

Des Weiteren müssen die Verfahren effizient sein und für große Ontologien anwendbar sein. Generell ist die Grundkomplexität die Kardinalität des kartesischen Produkts der Mengen der Entitäten der beiden Ontologien. Die Effizienz kann durch diverse Techniken gesteigert werden. Die Techniken lassen sich nach [6] klassifizieren:die Reduktion des Suchraums, die

Parallelisierung des Match- Workflows, Self-Tuning von Match- Workflows und die Wiederverwendung bereits berechneter Mappings. Bei der Reduktion des Suchraums wird die Anzahl der notwendigen Vergleiche reduziert. Eine Variante der Reduktion ist die Partitionierung wie es z.B. in [7] angewendet wird. Bei der Partitionierung werden die Entitäten der beiden Ontologien in Partitionen aufgeteilt. Es werden ausschließlich Entitäten verglichen, deren Partitionen sich ähnlich sind. Die Partitionen einer Ontologie sind durch strukturelles oder linguistisches Clustering ermittelbar.

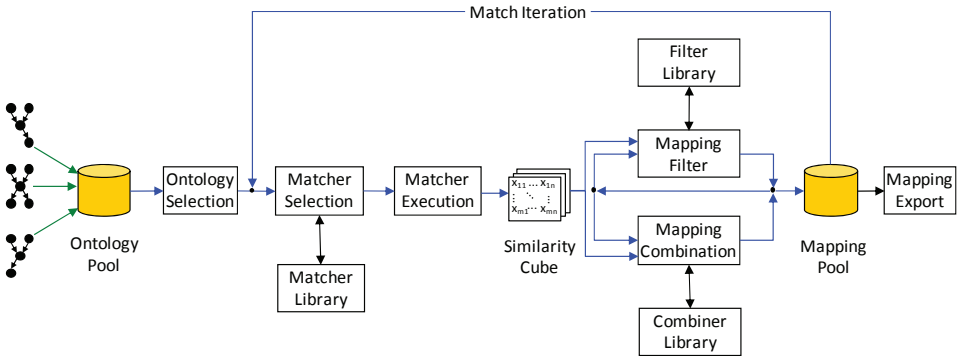
Bei der Parallelisierung wird der Match- Workflow auf mehreren Prozessoren oder Rechnern verteilt ausgeführt. Die Grundidee ist die Zerlegung des Match- Problems in kleinere Teilprobleme, die unabhängig voneinander gelöst werden können. Bei der Zerlegung des Problems kann z.B. eine Partitionierung der Ontologien angewendet werden.

Das Ziel des Self- Tuning ist die Identifikation einer guten Konfiguration für die jeweilige Machtaufgabe. Die meisten Verfahren verwenden Matcher oder Match- Workflows, die eine manuelle Konfiguration verlangen, wie z.B. die Spezifikation des Match- Workflows, die Spezifikation der Thresholds bzgl. der Ähnlichkeit eines Matchers usw. Die Konfiguration ist ein entscheidender Faktor für die Effizienz und Qualität eines Verfahrens. Um eine passende Konfigurationen zu bestimmen, ist es möglich durch semi- automatische Lernverfahren und bereits durchgeführte Matchprozesse eine Konfiguration zu ermitteln, die eine höhere Effizienz und eine bessere Qualität ermöglicht.

Ein weiterer Ansatz, der die Effizienz erhöht, ist die Wiederverwendung und Komposition von bereits bestehenden Mappings. Eine Voraussetzung für die Anwendung dieser Methode ist die Verwaltung der berechneten Mappings zwischen den Ontologien. Da sich Ontologien zu einer Thematik überlappen, können die verwalteten Mappings verwendet werden, um neue Mappings zwischen zwei Ontologien O_1 und O_2 zu berechnen, die bisher nicht miteinander verknüpft sind. Das neue Mapping wird durch eine *Intermediate* - Ontologie (IO) ermittelt, die die beiden Ontologien miteinander verknüpft.

Im Folgenden werden die Match- Verfahren GOMMA, ServOMap und YAM++ näher vorgestellt. Die generierten Mappings dieser Match- Verfahren werden im Kapitel 5 für die Evaluierung verwendet. GOMMA [8] (Generic Ontology Matching and Mapping Management) ist eine Infrastruktur, die für die Verwaltung von Ontologie- , Evolution- und Annotations- mappings konzipiert ist. Des Weiteren werden die Versionen der betrachteten Ontologien mithilfe eines generischen Datenmodells in einer relationalen Datenbank verwaltet. Die Infrastruktur realisiert drei Funktionskomponenten: *Match*, *DIFF* und *Evolution*. Die *Match* Funktionskomponente ist für die Bestimmung eines Ontologiealignments verantwortlich. Mithilfe dieser Komponente ist die *DIFF* Komponente realisierbar. Diese Komponente ermittelt die Menge von Änderungsoperationen, die notwendig sind, um eine alte Version einer Ontologie in die neue Version zu überführen. Um die Auswirkungen der Änderungsoperationen zu ermitteln, werden Verfahren verwendet, die eine Evolutionsanalyse durchführen. Diese Verfahren sind in der *Evolution* Komponente zusammengefasst und verwenden die Funktionalitäten der *DIFF* Komponente. Die *Match* Komponente basiert auf einem ähnlichen Ansatz der ebenfalls in COMA++

Abbildung 2.3: Ablauf des Matchverfahrens der GOMMA *Match*- Komponente(Quelle: [8])



[9] verfolgt wird. Jedoch skaliert GOMMA durch effiziente Techniken für große Ontologien, wie z.B. der Parallelisierung des Match- Workflows. Der prinzipielle Ablauf des Matchverfahrens ist in Abbildung 2.3 dargestellt. Dieser Ansatz basiert auf der Definition eines komplexen Match- Workflows, der sich aus einfachen Match- Verfahren, wie z.B. linguistische Match-Verfahren für spezifizierte Attribute, strukturelle Match- Verfahren, instanzbasierte Matcher, etc. zusammensetzt. Des Weiteren können bestehende Mappings für die Berechnung des Mappings verwendet werden. Die Ontologien, für die ein Mapping berechnet werden soll, müssen zu Beginn in das GOMMA- Repository importiert werden. Im zweiten Schritt erfolgt die Auswahl der Matcher aus der Matcher- Library und die Ausführung. Jeder Matcher berechnet für jedes Vergleichspaar einen Ähnlichkeitswert. Diese Werte werden in einem 3-dimensionalen Similarity-Cube gespeichert. Durch diese Datenstruktur kann nach der Ausführung der ausgewählten Matcher das Ergebnis ermittelt werden. Die Emittlung erfolgt durch spezifizierte Filter- und Selektionsstrategien sowie durch die Kombination der Ergebnisse der ausgewählten einfachen Matcher.

YAM++ [10] ist ein hybrides Match- System, das sich aus linguistischen und strukturellen Matchern zusammensetzt. Die linguistischen Matcher werden für die Generierung eines initialen Mappings verwendet. Diesbezüglich werden die Properties der Konzepte verwendet, um ein Alignment zu erzeugen. Die Matcher können unabhängig oder kombinierend verwendet werden. Die Wahl der zu verwendenden Matcher und die entsprechenden Metriken werden durch Verfahren des maschinellen Lernens ermittelt, wie z.B. Entscheidungsbäume, Support Vector Machines, NaiveBayes, etc. Die notwendige Trainingsmenge wird entweder von dem Anwender bereitgestellt oder aus den entsprechenden Wissensbasen extrahiert. Wenn keine Trainingsmenge zur Verfügung steht, werden Techniken des Information- Retrievals verwendet, um passende Metriken zu erzeugen. Das resultierende Alignment wird verwendet, um den strukturellen Matcher zu initialisieren. Dieser Matcher basiert auf dem Ansatz des Similarty-Flooding Algorithmus [11]. Bei diesem Algorithmus werden die existierenden Ähnlichkeiten durch eine Graphdatenstruktur propagiert, so dass weitere Korrespondenzen ermittelt werden. Im letzten Schritt werden die Korrespondenzen auf logische Inkonsistenzen überprüft und gegebenfalls inkorrekte Korrespondenzen entfernt.

ServOMap [12] ist eine Funktionskomponente innerhalb des ServO OR Systems, das für die Verwaltung heterogener Ontologien konzipiert ist. Das System ähnelt einem Information Retrieval System. Eine Ontologie wird in ServO OR als Korpus aufgefasst und die Konzepte und Relationen als Dokumente. Prinzipiell basiert die Identifikation eines Alignments auf der Berechnung der Ähnlichkeiten zwischen den Vektoren der Terme, die die Entitäten der Ontologien repräsentieren. Bei der Erstellung der Vektoren werden für jede Entität eine Menge von Vorverarbeitungsschritten abgearbeitet, wie z.B. die Entfernung von Stoppwörtern, Kleinschreibung, Wortkonkatenierung, etc. Die Schritte sind abhängig von der Menge der Features, die eine Entität beinhaltet. Neben dem linguistischen Matcher wird ebenfalls ein struktureller Matcher nach der Prozessierung des linguistischen Matchers verwendet. Dieser Matcher definiert zwei Konzepte als gleich, wenn die Kontexte der beiden Konzepte ähnlich sind. Der Kontext eines Konzept entspricht den Superkonzepten, Subkonzepten und den Schwesterkonzepten.

2.3 Reasoning- Verfahren

Reasoning- Verfahren werden verwendet, um die Konsistenz einer Wissensbasis zu überprüfen oder implizite Subkonzeptbeziehungen abzuleiten. Mithilfe der Konsistenzprüfung ist es möglich, implizite Axiome abzuleiten und neue Wissensbasen zu generieren. Aufgrund der Verfahren ist es nicht notwendig alle Axiome einer Wissensbasis explizit zu codieren. Es wird zwischen vollständigen Reasonern und unvollständigen Reasonern differenziert. Die vollständigen Reasoner, wie z.B. HermiT und Pellet, betrachten alle Konstruktionsmöglichkeiten der zugrundeliegenden Beschreibungslogiksprache. Im Gegensatz dazu, verwenden unvollständige Reasoner, wie z.B. der ELK Reasoner, eine Teilmenge der Konstruktionsoperatoren, um die Komplexität einzugrenzen. Im Folgenden wird das Tableau Verfahren dargestellt [5] und Optimierungsmöglichkeiten präsentiert wie sie in HermiT realisiert sind. Des Weiteren wird der ELK- Reasoner erläutert, der implizite Subkonzeptbeziehungen identifiziert.

2.3.1 Tableaubasierte Verfahren

Eine typische Methode für die Überprüfung der Konsistenz einer Wissensbasis ist das Tableau-Verfahren. Generell überprüft das Tableau- Verfahren, die Erfüllbarkeit einer Wissensbasis durch die Ableitung einer generischen Interpretation mithilfe von Ableitungsregeln. Um ein implizites Axiom der Wissensbasis abzuleiten, wird durch das Verfahren bewiesen, dass die Negation des Axioms bzgl. der WB unerfüllbar ist. Um den Tableaubeweiser anwenden zu können, müssen alle Axiome der TBox in Negationsnormalform (NNF) transformiert werden. Bei der NNF ist das Vorkommen von Negationen ausschließlich vor atomaren Konzepten möglich. Generell wird die Unerfüllbarkeit einer Wissensbasis durch das Zeigen der Abgeschlossenheit eines Tableaus bewiesen. Ein Tableau ist ein Baum deren Knoten aus Zuordnungen von Individuen zu Axiomen besteht. Ein Pfad des Baumes repräsentiert eine ABox, die die Zuordnung der generischen Individuen zu Konzepten, Relationen oder Axiomen beinhaltet. Ein Tableau ist abgeschlossen, wenn das Tableau nicht mehr erweitert werden kann und ein Widerspruch in allen abgeleiteten ABoxen existiert. Ein Widerspruch in einer ABox existiert, wenn eine Zuordnung eines atomaren Konzepts zu einem Individuum sowie deren Negation existiert, dass

heißt $C(a)$ und $\neg C(a)$ sind in einem Pfad des $Tableau$. Des Weiteren existiert ein Widerspruch, wenn die Axiome $\geq m\ R(a)$ und $\leq n\ R(a)$ in der ABox sind und $n < m$ gilt. Die Initialisierung des Verfahrens umfasst die Instanziierung eines beliebigen Axioms der Wissensbasis mit einer generischen Instanz a. Das Tableau wird durch Regeln erweitert, die eine Vorbedingung und eine Menge von inferierten Axiomen umfassen. Eine Übersicht der Regeln ist in Tabelle 2.2 dargestellt. Technisch wird das Tableau durch einen Backtracking Ansatz erzeugt, indem eine aktuelle ABox konstruiert wird. Ein bestehender Pfad ist erweiterbar, wenn eine Erweiterungsregel anwendbar ist und die abgeleiteten Axiome nicht bereits im Tableau existieren. Wenn der aktuelle Pfad nicht mehr erweiterbar ist und kein Widerspruch existiert, ist die konstruierte ABox ein Modell der Wissensbasis. Wenn ein Widerspruch existiert, wird das Verfahren an dem Knoten fortgesetzt, der eine alternative Belegung für das Individuum festlegt. Eine alternative Belegung entsteht durch Axiome der Form $(C \sqcup D)(a)$.

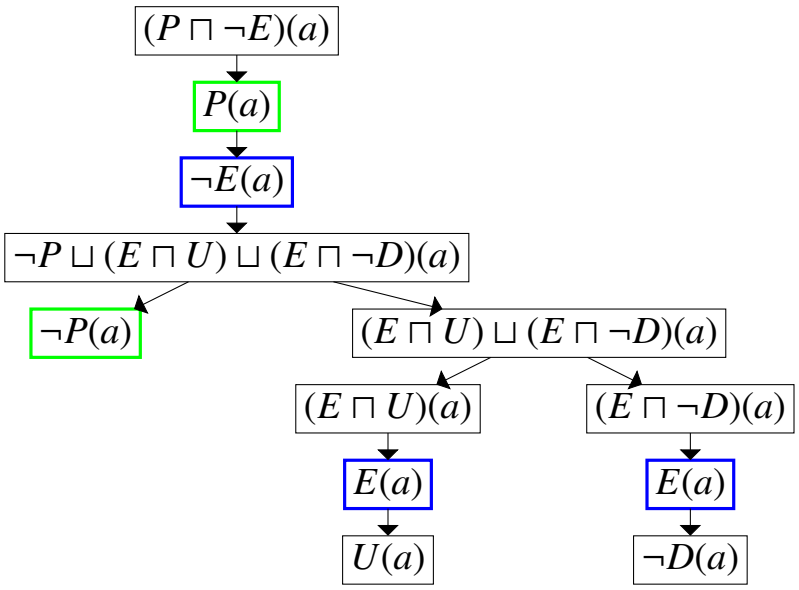
Nr.	Prämisse	Aktion
1.	$C(a) \in W(ABox)$	Hinzufügen von $C(a)$
2.	$R(a,b) \in W(ABox)$	Hinzufügen von $R(a,b)$
3.	$C \in W$	Hinzufügen von $C(a)$ für ein neues Individuum a
4.	$(C \sqcup D)(a) \in Tableau$	Splitten des Pfades, Hinzufügen von $C(a)$ in 1. Pfad und $D(a)$ in 2. Pfad
5.	$(C \sqcap D)(a) \in Tableau$	sequentielles Hinzufügen von $C(a)$ und $D(a)$
6.	$(\exists R.C)(a) \in Tableau$	Hinzufügen von $R(a,b)$ und $C(b)$ für ein neues Individuum b
7.	$(\forall R.C)(a) \in Tableau$	Hinzufügen der Einschränkung $C(b)$ für ein bekanntes Individuum b, falls $R(a,b) \in Tableau$
8.	$\geq nR.C(a)$	Hinzufügen neuer Individuen $C(b_1), ...C(b_n)$ für die gilt:$(a,b_i) \in R^I (1 \leq i \leq n)$, falls $C(b_1), ...C(b_n)$ nicht existieren
9.	$\leq nR.C(a)$	Wenn Individuen $C(b_1), ...(C(b_{n+1}))$ existieren mit $(a,i) \in R^I (1 \leq i \leq n+1)$, dann verschmelze Individuen durch Ersetzen von b_i durch b_j

Tabelle 2.2: Erweiterungsregeln für das Tableau Verfahren

Die Abbildung 2.4 veranschaulicht das Verfahren an einem Beispiel. Es ist folgendes Axiom der Wissensbasis gegeben: $\neg P \sqcup (E \sqcap U) \sqcup (E \sqcap \neg D)$. Die Anfrage ist: $P \sqsubseteq E = \neg P \sqcup E$. Die Negation von dem Axiom ist: $P \sqcap \neg E$. Im ersten Schritt wird die Regel 1. auf das negierte Axiom angewendet. Daraus folgen die Axiome $P(a)$ und $\neg E(a)$, die als sequentieller Pfad an den bestehenden Pfad fortgesetzt werden. Im zweiten Schritt wird das Axiom der Wissensbasis hinzugefügt. Durch die Anwendung der Regel 4. wird der bestehende Pfad des Axioms $(\neg P \sqcup (E \sqcap U) \sqcup (E \sqcap \neg D))(a)$ gesplittet in $\neg P(a)$ und $(E \sqcap U) \sqcup (E \sqcap \neg D)(a)$. Durch weitere Anwendung der Regel 4. wird der Pfad mit dem Axiom $((E \sqcap U) \sqcup (E \sqcap \neg D))(a)$ gesplittet in $(E \sqcap U)(a)$ und $(E \sqcap \neg D)(a)$. Durch die Anwendung der Regel 5. auf die beiden Axiome entstehen die Teilpfade $E(a)$ und $U(a)$ bzw. $E(a)$ und $\neg D(a)$. Alle Pfade sind abgeschlossen, da der

erste Pfad $P(a)$ und $\neg P(a)$ enthält und die anderen beiden Pfade $E(a)$ und $\neg E(a)$. Demzufolge ist $P \sqcap \neg E$ nicht erfüllbar und somit folgt: $P \sqsubseteq E$ ist herleitbar.

Abbildung 2.4: Beispiel eines Tableaus



Das standardisierte Verfahren ist u.a. aufgrund der Disjunktion nicht effizient. Die Regel für die Disjunktion und somit ebenfalls für die Subsumption verursachen eine Überprüfung von zwei ABoxen. Da die Subsumption ein essentieller Bestandteil der Axiome in Ontologien ist, führt eine ungünstige Konstruktion des Tableaus zu einer hohen Menge an zu konstruierenden ABoxen. Eine übliche Optimierungsstrategie ist die Verwendung von sogenannten Absorptionstechniken. Diese Techniken reduzieren die Anzahl der zu expandierenden Pfade und somit die Anzahl der zu überprüfenden ABoxen.
Der Basisabsorptionsalgorithmus versucht die Axiome der TBox in die Form $B \sqsubseteq C$ zu transformieren, wobei B atomar ist. Die Subsumption repräsentiert eine Disjunktion $\neg B \sqcup C$. In der Regel muss sowohl $\neg B$ und auch C für jedes Individuum der aktuellen ABox überprüft werden. Der Absorptionsalgorithmus überprüft jedoch $C(a)$ für jedes Individuum der ABox ausschließlich, wenn $B(a)$ in der ABox existiert. Dieser Algorithmus ist die Basis für weitere Absorptionstechniken wie z.B. die binäre Absorption [13]. Diese Technik transformiert die Axiome in die Form $B_1 \sqcap B_2 \sqsubseteq C$. $C(a)$ wird ausschließlich abgeleitet, wenn $B_1(a)$ und $B_2(a)$ in der ABox existieren.
Jedoch sind diese Techniken nicht ausreichend, um eine gute Effizienz zu gewährleisten. Der HermiT- Reasoner von [14] generalisiert die existierende Menge von Absorptionstechniken, so dass diese simultan ausführbar sind. Des Weiteren werden zusätzliche Absorptionstechniken eingeführt, die ausschließlich im Hypertableau Verfahren anwendbar sind.

Ein weitere Ursache für die Ineffizienz ist die Einführung neuer Individuen durch die existentielle Quantifizierung (Regel 6. in 2.2). Durch diese Erweiterungsregeln können Zyklen generiert werden, so dass naive tableaubasierte Implementierungen nicht terminieren. Sei z.B.

das Axiom $\neg Person \sqcup \exists hatEltern.Person$ gegeben. Bei der Überprüfung dieses Axioms wird immer eine neue Instanz für das Konzept *Person* durch die Regel 6 erzeugt. Dadurch wird das Tableau endlos erweitert. Generell werden solche Zyklen durch Blocking- Verfahren verhindert, indem die Erweiterung der ABox abgebrochen wird, wenn zwei Individuen bei der Erweiterung der ABox aufeinander folgen und zu den selben Konzepten gehören. Jedoch ist diese Art des Blockings nicht effizient, da ausschließlich aufeinanderfolgende Individuen betrachtet werden. HermiT realisiert eine Blockingtechnik, bei der fast jedes Individuum ein anderes Individuum blocken kann.

2.3.2 Schlussfolgern von impliziten Subklassenbeziehungen

Das Schlussfolgern von impliziten Subsumptionen in einer Ontologie bezeichnet man als Klassifizierung. Durch die Klassifizierung einer Ontologie kann ein höherer Grad der Strukturierung erreicht werden oder präzisere Anfrageergebnisse bzgl. der Anfrage aller Instanzen eines bestimmten Konzepts realisiert werden. Die Klassifizierung wird ebenfalls für das Verfahren dieser Arbeit verwendet, da diese Methode z.T. auf der *is_a* Hierarchie basiert.
Die meisten Klassifizierungsverfahren verwenden abgeschwächte Beschreibungssprachen wie z.B. $\mathcal{ELH}$ in [15] oder $\mathcal{EL}$ beim ELK- Reasoner [16]. $\mathcal{EL}$ beinhaltet die Subsumption von Konzepten, aber keine Definition der ABox. Ein Konzept kann atomar sein, durch eine Konjunktion ($C \sqcap D$) oder durch eine existentielle Quantifizierung ($\exists R.C$) definiert werden. Prinzipiell verwendet ein Klassifizierungsverfahren eine Menge von Inferenzregeln, die aus einer Prämisse, gegebenfalls Nebenbedingungen und einer Folgerung bestehen. Wenn eine Prämisse für eine bestimmte Regel erfüllt ist, kann das gefolgerte Axiom für weitere Inferenzen verwendet werden. Beispielhaft wird im Folgenden der ELK- Klassifizierer beschrieben.
Eine Teilmenge der Inferenzregeln, die der Reasoner verwendet, ist in der Abbildung 2.5 dargestellt. Der Ausdruck $init(C)$ ist eine Initialisierungsfunktion für die Ableitung der Superkonzepte

Abbildung 2.5: Teilmenge der Inferenzregeln für die Klassifizierung einer Ontologie mittels des ELK Reasoner

$$R_0: \frac{init(C)}{\overline{C} \sqsubseteq C} \qquad R^+_\top : \frac{init(C)}{\overline{C} \sqsubseteq \top}\ ,\ \top \text{ kommt negativ in } \mathcal{O} \text{ vor.} \qquad R_\sqsubseteq: \frac{\overline{C} \sqsubseteq D}{\overline{C} \sqsubseteq E},\ D \sqsubseteq E \in \mathcal{O}$$

$$R^-_\sqcap: \frac{\overline{C} \sqsubseteq D_1 \sqcap D_2}{\overline{C} \sqsubseteq D_1 \quad \overline{C} \sqsubseteq D_2} \qquad R^+_\sqcap: \frac{\overline{C} \sqsubseteq D_1 \quad \overline{C} \sqsubseteq D_2}{\overline{C} \sqsubseteq D_1 \sqcap D_2},\ D_1 \sqcap D_2 \text{ kommt negativ in } \mathcal{O} \text{ vor.}$$

$$R^-_\exists: \frac{\overline{C} \sqsubseteq \exists R.D}{init(D) \quad C \sqsubseteq \exists R.\overline{D}} \qquad R^+_\exists: \frac{D \sqsubseteq \exists R.\overline{C} \quad \overline{C} \sqsubseteq E}{\overline{D} \sqsubseteq \exists R.E},\ \exists R.E \text{ kommt negativ in } \mathcal{O} \text{ vor.}$$

von C. Die Schreibweise $\overline{C}$ repräsentiert den Kontext eines Konzepts. Mithilfe des Kontexts ist eine effiziente Berechnung der deduktiven Hülle realisierbar. Des Weiteren erscheint ein Konzept C *negativ* (resp. *positiv*), wenn es ein Teilausdruck eines Konzepts D (resp. E) ist für mindestens ein Axiom $D \sqsubseteq E$. Das Verfahren ist in drei Phasen unterteilt: Indexierung, Berechnung der deduktiven Hülle und der Konstruktion der Taxonomie.
Mithilfe der Indexierung ist eine effiziente Identifikation der Axiome bzgl. eines Konzepts

realisierbar, die für eine Nebenbedingung einer Inferenzregel verwendet werden können. Für jedes Konzept werden drei Mengen spezifiziert: $C.toldSups$, $C.negConj$, $C.negExis$. Für eine Relation R wird die Menge $R.negExis$ definiert.

$$C.toldSups = \{D | C \sqsubseteq D\}$$
$$C.negConj = \{< D, C \sqcap D > | (C \sqcap D) \text{ kommt negativ in } \mathcal{O} \text{ vor}\} \cup$$
$$\{< D, D \sqcap C > | (C \sqcap D) \text{ kommt negativ in } \mathcal{O} \text{ vor}\}$$
$$C.negExis = \{< R, \exists R.C > | \exists R.C \text{ kommt negativ in } \mathcal{O} \text{ vor}\}$$
$$R.negExis = \{< C, \exists R.C > | \exists R.C \text{ kommt negativ in } \mathcal{O} \text{ vor}\}$$

$C.toldSups$ enthält alle Superkonzepte von C. Die Axiome dieser Menge können für die Regel $R_{\sqsubseteq}$ für ein Konzept C verwendet werden. $C.negConj$ ist eine Menge für ein Konzept C von Schlüssel- Wert Paaren, wobei der Wert die Konjunktion von zwei Konzepten darstellt. Der Schlüssel ist das Konzept D. Die Konzepte dieser Menge werden für die Nebenbedingung der Regel $R^+_{\sqcap}$ benötigt. $C.negExis$ ist eine Menge für ein Konzept C von Schlüssel- Wert Paaren, wobei der Wert eine existentielle Quantifizierung ist($\exists R.C$) und der Schlüssel die Relation. Die Menge $R.negExis$ ist ähnlich wie die $C.negExis$ mit dem Unterschied das die Menge für jede Relation definiert wird und der Schlüssel folglich das Konzept ist. Die letzten beiden Mengen enthalten die Konzepte, die für die Regel $R^+_{\exists}$ verwendet werden. Zur Veranschaulichung wird ein kleines Beispiel dargestellt.
Es seien folgende Axiome gegeben:

$$A \sqsubseteq \exists R.(B \sqcap C)$$
$$A \sqcap \exists R.B \sqsubseteq C$$

Die Indexmengen für die einzelnen Konzepte sind:

$$A.toldSups = \{\exists R.(B \sqcap C)\} \quad A.negConj = \{< \exists R.B, A \sqcap \exists R.B >\}$$
$$B.negExis = \{< R, \exists R.B >\} \quad (A \sqcap \exists R.B).toldSups = \{C\}$$
$$(\exists R.B).negConj = \{< A, A \sqcap \exists R.B >\} \quad R.negExis = \{< B, \exists R.B >\}$$

Die Berechnung der deduktiven Hülle basiert auf der Anwendung der vorab definierten Inferenzregeln. Um die Effektivität zu steigern, wird der Suchraum der Axiome, die für eine Regel benötigt werden, verringert. Die Reduzierung des Suchraums beruht auf der Einführung eines Kontexts für jedes Konzept. Der Kontext eines Konzepts $\overline{C}$ beinhaltet zwei Queue Datenstrukturen, die die Axiome beinhalten, die das jeweilige Konzept enthalten. Eine Queue (`processedQueue`) enthält alle prozessierten Axiome und die zweite Queue (`localTODOQueue`) alle noch zu verarbeitenden Axiome. Global wird eine Queue der Kontexte definiert, die alle aktiven Kontexte enthält. Die `localTODOQueue` wird durch die Regel $init(C)$ für ein Konzept C initialisiert. Des Weiteren ermöglicht die Separierung der Axiome in konzeptbezogene Kontexte die Parallelisierung der Berechnung. Ein Arbeitsprozess kann unabhängig von allen anderen Arbeitsprozessen einen Kontext verarbeiten. Neben den beiden Queue Datenstrukturen werden für einen Kontext zwei Mengen verwaltet, die Axiome beinhalten, die als Prämisse bei einem Konzept C fungieren können.

$$C.superConcepts = \{D | \overline{C} \sqsubseteq D \text{ wurde verarbeitet}\}$$
$$C.predecessors = \{< R, D > | D \sqsubseteq R.\overline{C} \text{ wurde verarbeitet}\}$$

Prozedural verarbeitet ein Prozess einen aktiven Kontext aus der globalen Queue der Kontexte, indem er aus der `contextQueue` den ersten Kontext wählt und die Axiome der `localTODOQueue` sukzessive abarbeitet. Ein Axiom wird aus der `localTODOQueue` entfernt. Wenn es nicht in der `processedQueue` vorhanden ist, werden mithilfe der Regeln alle Axiome berechnet, die mithilfe des Axioms inferiert werden können. Das aktuelle Axiom wird zu der `processedQueue` hinzugefügt und alle inferierten Axiome und Axiome der `processedQueue` werden zu der `localTODOQueue` hinzugefügt. Ein Kontext $\overline{C}$ ist inaktiv, wenn die `localTODOQueue` leer ist. Ein leerer Kontext wird aus der `contextQueue` gelöscht. Die Berechnung terminiert, wenn die `contextQueue` leer ist.

Da die Berechnung die vollständige transitive Hülle der Subsumptionen ermittelt, aber lediglich die direkten Superklassen eines Konzepts für eine Taxonomie verwendet werden, müssen die indirekten Superkonzepte für ein Konzept entfernt werden.

Zur Veranschaulichung wird das Beispiel fortgesetzt. Das Ziel ist die Identifikation aller Superkonzepte des Konzepts A. Die Tabelle 2.3 zeigt den Ablauf der Berechnung der deduktiven Hülle für das Konzept A. Der erste Schritt ist die Ableitung der Initialisierungsfunktion. Im zweiten Schritt wird das Axiom $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$ durch die Regel $R \sqsubseteq$ mithilfe der Axiome $\overline{A} \sqsubseteq A$ und $\exists R.(B \sqcap C)$ der Indexmenge $A.toldSups$ abgeleitet. Durch die Regel $R^{-}_{\exists}$ wird im Schritt 3. der Kontext $\overline{B \sqcap C}$ mit den Axiomen $init(B \sqcap C)$ und $\exists A \sqsubseteq \exists R.(\overline{B \sqcap C})$ initialisiert. Durch die abgeleiteten Axiome von Schritt 3. bis 6. im Kontext $\overline{B \sqcap C}$ wird erneut der Kontext $\overline{A}$ mit dem Axiom $A \sqsubseteq \exists R.B$ im Schritt 7. aktiviert. Mithilfe dieses Axioms und des Axioms $\overline{A} \sqsubseteq A$ wird das Axiom $\overline{A} \sqsubseteq A \sqcap \exists R.B$ im Schritt 8. durch die Regel $R^{+}_{\sqcap}$ abgeleitet. Des Weiteren wird der Kontext $\overline{B}$ initialisiert durch die Regel $R^{-}_{\exists}$. Im Schritt 9. wird die Initialisierungsfunktion des Kontext $\overline{B}$ abgeleitet und im Kontext $\overline{A}$ lediglich die Initialisierungsfunktion aus der `localTODOQueue` gelöscht. Die Schritte 10. und 11. umfassen lediglich eine Löschung der Axiome aus der `localTODOQueue` im Kontext $\overline{A}$. Im Schritt 12. wird durch das Axiom $\overline{A} \sqsubseteq A \sqcap \exists R.B$ und durch das Axioms der Indexmenge $(A \sqcap \exists R.B).toldSups$ das Ergebnis der Berechnung abgeleitet: $\overline{A} \sqsubseteq C$.

Tabelle 2.3: Ablauf der Berechnung der deduktiven Hülle für das Konzept A

Kon.	$\overline{A}$		$\overline{B \sqcap C}$		$\overline{B}$		
Schr.	*todo*	*processed*	*todo*	*processed*	*todo*	*processed*	Regel
0	$init(A)$						
1	$\overline{A} \sqsubseteq A$	$init(A)$					$R_0(\overline{A})$
2	$\overline{A} \sqsubseteq \exists R.(B \sqcap C)$ $init(A)$	$init(A)$ $\overline{A} \sqsubseteq A$					$R_{\sqsubseteq}(\overline{A})$
3	$init(A)$	$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$	$init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$				$R^-_{\exists}(\overline{A})$
4		$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$	$A \sqsubseteq \exists R.(\overline{B \sqcap C})$ $\overline{B \sqcap C} \sqsubseteq B \sqcap C$	$init(B \sqcap C)$			$R_0(\overline{B \sqcap C})$
5		$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$	$\overline{B \sqcap C} \sqsubseteq B \sqcap C$	$init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$			
6		$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$	$(\overline{B \sqcap C}) \sqsubseteq B$ $(\overline{B \sqcap C}) \sqsubseteq C$ $init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$	$init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$ $\overline{B \sqcap C} \sqsubseteq B \sqcap C$			$R^-_{\sqcap}(\overline{B \sqcap C})$
7	$\overline{A} \sqsubseteq \exists R.B$ $init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$	$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$	$(\overline{B \sqcap C}) \sqsubseteq C$ $init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$	$init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$ $\overline{B \sqcap C} \sqsubseteq B \sqcap C$ $(\overline{B \sqcap C}) \sqsubseteq B$			$R^+_{\exists}(\overline{B \sqcap C})$
8	$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$ $\overline{A} \sqsubseteq A \sqcap \exists R.B$	$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$ $\overline{A} \sqsubseteq \exists R.B$	$init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$	$init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$ $\overline{B \sqcap C} \sqsubseteq B \sqcap C$ $(\overline{B \sqcap C}) \sqsubseteq B$ $(\overline{B \sqcap C}) \sqsubseteq C$	$init(B)$ $A \sqsubseteq \exists R.B$		$R^+_{\sqcap}(\overline{A})$, $R^-_{\exists}(\overline{A})$
9	$\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$ $\overline{A} \sqsubseteq A \sqcap \exists R.B$	$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$ $\overline{A} \sqsubseteq \exists R.B$	$A \sqsubseteq \exists R.(\overline{B \sqcap C})$	$init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$ $\overline{B \sqcap C} \sqsubseteq B \sqcap C$ $(\overline{B \sqcap C}) \sqsubseteq B$ $(\overline{B \sqcap C}) \sqsubseteq C$	$\overline{B} \sqsubseteq B$ $A \sqsubseteq \exists R.\overline{B}$	$init(B)$	$R_0(\overline{B})$
12	$\overline{A} \sqsubseteq C$	$init(A)$ $\overline{A} \sqsubseteq A$ $\overline{A} \sqsubseteq \exists R.(B \sqcap C)$ $\overline{A} \sqsubseteq \exists R.B$ $\overline{A} \sqsubseteq A \sqcap \exists R.B$		$init(B \sqcap C)$ $A \sqsubseteq \exists R.(\overline{B \sqcap C})$ $\overline{B \sqcap C} \sqsubseteq B \sqcap C$ $(\overline{B \sqcap C}) \sqsubseteq B$ $(\overline{B \sqcap C}) \sqsubseteq C$		$init(B)$ $\overline{B} \sqsubseteq B$ $A \sqsubseteq \exists R.\overline{B}$	$R_{\sqsubseteq}(\overline{A})$

Kapitel 3

Map- Repair Verfahren für die Reparatur eines Ontologie Alignments

3.1 Einführung

Ein Alignment zwischen zwei Ontologien umfasst semantische Korrespondenzen zwischen den Konzepten der Ontologien. Die semantischen Korrespondenzen können Subsumptionen und Äquivalenzen sein. Durch ein Alignment wird die Subsumptionsstruktur der integrierten WB verändert, so dass unerfüllbare Konzepte generiert werden können. Für die Identifikation der logisch inkorrekten Korrespondenzen verwenden Map- Repair Verfahren eine vereinfachte integrierte WB der Form $O_1 \cup O_2 \cup \mathcal{M}$.

Das Ziel der Map- Repair Verfahren ist die Ermittlung einer Menge von Korrespondenzen, auch Diagnose Δ genannt, so dass beim Löschen dieser Korrespondenzen aus dem ursprünglichen Alignment eine kohärente Wissensbasis $O_1 \cup O_2 \cup \mathcal{M} \backslash \Delta$ entsteht. Des Weiteren soll Δ die kleinste Menge von Korrespondenzen sein, so dass jede kleinere Menge zu einer inkohärenten WB führt. Idealerweise erhöht sich durch das Löschen inkorrekter Korrespondenzen die Precision und der Recall bleibt gleich. In der Praxis ist aber eher eine Reduzierung des Recalls zu beobachten, wobei sich die Erhöhung der Precision und die Senkung des Recalls bzgl. des F- Measure nivellieren. Da beim Ontologie- Matching große Ontologien betrachtet werden, ist es notwendig effiziente Reparaturverfahren zu entwickeln, damit diese als Post-Processing Schritt für bestehende Match- Verfahren verwendet werden können.

Ein naiver Ansatz ist die Betrachtung aller Teilmengen Δ' des originalen Mappings in absteigender Reihenfolge bzgl. der Anzahl der Korrespondenzen und die Überprüfung mittels eines vollständigen Reasoner, ob eine kohärente Wissensbasis $O_1 \cup O_2 \cup \mathcal{M} \backslash \Delta'$ entsteht. Bei der Betrachtung aller Teilmengen muss ein vollständiger Reasoner $2^{|\mathcal{M}|}$ Überprüfungen durchführen. Jedoch ist dies aufgrund der geringen Effizienz der existierenden vollständigen Reasoner nicht realisierbar.

Aus diesem Grund verwenden die Verfahren hauptsächlich keine vollständigen Reasoner oder in reduzierter Form für die Überprüfung der Kohärenz der Ontologie. Des Weiteren fokussieren sich die Verfahren auf eine Teilmenge der $\mathcal{SHOIN}(D)$ Beschreibungslogik. Grundsätzlich werden Disjunktheitsbeziehungen und Subsumptionen betrachtet. Es kann aufgrund der Einschränkung und ohne den Einsatz von vollständigen Reasonern nicht garantiert werden, dass die identifizierte Menge von Korrespondenzen minimal ist und zu einer kohärenten WB führt. Man bezeichnet diese Menge deshalb als Reparaturmenge $R^{\approx}$. Bisher existieren nur wenige Map- Repair Verfahren. Im Folgenden werden die drei existierenden Verfahren beschrieben, deren Ziel die Reparatur eines gegebenen Alignments für zwei Ontologien ist. Anschließend wird im Kapitel 4 das konzipierte und implementierte Verfahren vorgestellt. Im Kapitel 5 werden die Verfahren vergleichend evaluiert.

3.2 LogMap

LogMap[17] ist ein Reparaturverfahren, das eine autarke Realisierung der Funktionskomponente des LogMap Match- Verfahrens ist. LogMap betrachtet ausschließlich die Subsumption und

Disjunktheit von Konzepten. Die Eingabe umfasst zwei Ontologien O_1, O_2 und ein Mapping $\mathcal{M}$. Die Ausgabe ist die Menge der zu löschenden Korrespondenzen $\mathcal{R}^{\approx}$ und das modifizierte Mapping $\mathcal{M}$'. Es umfasst eine Einlesephase der Ontologie und die Berechnung der Reparaturmenge. Das Verfahren ist im Algorithmus 1 dargestellt. LogMap verwendet für die Überprüfung der

Algorithmus 1: LogMap- Repair Verfahren

```
   input  : O1, O2, M
   output: R≈, M'
 1 R≈ ← ∅
 2 M' ← M
 3 < P1P2 > ← encode(O1, O2)
 4 foreach C ∈ orderedVariables(P1 ∪ P2) do
 5     PC ← P1 ∪ P2 ∪ M' ∪ {C ← true}
 6     < satisfiable, M⊥ > ← DowlingGallier(PC)
 7     if ¬satisfiable then
 8         Rep ← ∅
 9         repSize ← 1
10         repeat
11             foreach RC ∈ M⊥ of size repSize do
12                 satisfiable ← DowlingGallier(PC\RC)
13                 if satisfiable then
14                     Rep ← Rep ∪ RC
15                 repSize ← repSize + 1
16         until Rep ≠ ∅
17         RC ← set with smallest aggregated confidence of Rep
18         M' ← M'\RC
19         R≈ ← R≈ ∪ RC
20 return <R≈, M' >
```

Erfüllbarkeit der Konzepte den Dowling- Gallier Algorithmus [18], der eine Hornformel als Eingabe verwendet. Diesbezüglich werden in der Einlesephase alle Subsumptions- und Disjunktheitsaxiome der beiden Ontologien in eine Hornformel P_1 und P_2 transformiert (Zeile 3). Eine Hornformel ist eine Konjunktion von Disjunktionen, wobei die Disjunktion höchstens ein positives Literal enthält. Die Axiome werden mittels folgender Regeln transformiert:

1. $A \sqsubseteq B : A \rightarrow B = \neg A \vee B$
2. $A \sqcap B \sqsubseteq \emptyset : A \wedge B \rightarrow false = \neg A \vee \neg B \vee false$
3. $A_1 \sqcap A_2 \sqsubseteq B : A_1 \wedge A_2 \rightarrow B = \neg A_1 \vee \neg A_2 \vee B$

Die Berechnung der Reparaturmenge erfolgt iterativ, indem die Erfüllbarkeit der logischen Formel $P_C = P_1 \cup P_2 \cup \mathcal{M}'$ für die Erfüllbarkeit aller Konzepte überprüft wird (Zeile 4-19). $\mathcal{M}'$ ist das modifizierte Alignment aus dem Korrespondenzen bereits entfernt sind. Die Reihenfolge

der Überprüfung der Konzepte wird durch die *is_a* Hierarchie der beiden Ontologien spezifiziert. Ein Konzept C wird vor einem Konzept D überprüft, wenn $C \sqsubseteq D$ gilt. Die Reihenfolge der Abarbeitung der Variablen, die die Formel $P_1 \cup P_2$ enthält, wird durch die Methode `orderedVariables` realisiert. Die Variable C, die das aktuelle Konzept repräsentiert wird als wahr in der Formel P_C ausgewertet (Zeile 5). Wenn die Formel P_C erfüllbar ist, wird mit der Variable des nächsten Konzepts fortgefahren. Wenn die Formel nicht erfüllbar ist, ist das aktuelle Konzept unerfüllbar (Zeile 7). Das Ergebnis der implementierten Version des Dowling-Gallier Algorithmus ist eine Variable vom Typ *boolean* und eine Obermenge $M_\perp$ von Korrespondenzen, die bei Unerfüllbarkeit eine potentielle Ursache darstellen. Diese Obermenge wird verwendet, um die kleinste Menge an Korrespondenzen zu identifizieren (Zeile 11-16), so dass P_C erfüllbar ist. Bei der Überprüfung wird eine Teilmenge der Korrespondenzen R_C von $M_\perp$ aus der logischen Formel P_C entfernt. Wenn die Formel erfüllbar ist (Zeile 13) wird die Teilmenge zur Reparaturmenge Rep für das Konzept C hinzugefügt. Die Identifikation der Teilmenge R_C erfolgt in aufsteigender Reihenfolge bzgl. der Kardinalität der Menge R_C. Die Ermittlung der Reparaturmenge Rep ist beendet, wenn diese Menge nicht mehr leer ist (Zeile 16). Wenn die Reparaturmenge Rep mehrere Mengen beinhaltet, wird die Menge R_C gewählt, die die geringste Konfidenz aggregiert (Zeile 17). Diese Menge wird aus dem bestehenden Alignment $\mathcal{M}'$ entfernt und zu der bereits bestehenden Reparaturmenge $\mathcal{R}^{\approx}$ hinzugefügt. Das Verfahren terminiert, wenn alle Konzepte verarbeitet sind.

3.3 Alcomo

Alcomo [19] ist ein musterbasiertes Reparaturverfahren. Dieser Ansatz ist unvollständig bzgl. der Kohärenz der WB $O_1 \cup O_2 \cup \mathcal{M}$. Um die Kohärenz zu gewährleisten, ist es möglich einen vollständigen Reasoner mit zu verwenden. Das Verfahren versucht ein Minimum bzgl. der aggregierten Konfidenz der Korrespondenzen der Reparaturmenge zu identifizieren. Die Aggregation der Konfidenz entspricht der Summe der Konfidenzen der Korrespondenzen, die die Reparaturmenge enthält. Diesbezüglich wird zwischen einer lokalen Diagnose und einer globalen Diagnose differenziert. Die lokale Diagnose wählt in jedem Berechnungsschritt bei der Ermittlung der Diagnose, die Korrespondenz mit der geringsten Konfidenz. Somit ist es möglich, dass die identifizierte Menge nicht das globale Minimum darstellt. Die globale Diagnose wird mithilfe des A* Algorithmus ermittelt. Der A* Algorithmus verwendet eine Baumdatenstruktur, die die bereits gefundenen Teillösungen verwaltet. Jede Teillösung $\mathcal{R}'$ aggregiert eine Konfidenz, die sich aus den Konfidenzen der Korrespondenzen der Teillösung zusammensetzt. Da eine Expansion des gesamten Baumes ineffizient ist, wird die Menge der noch zu löschenden Korrespondenzen heuristisch abgeschätzt. Der Algorithmus expandiert die Teillösung, mit der niedrigsten Summe der aggregierten und heuristischen Konfidenzen $g(\mathcal{R}')$ bzw. $h(\mathcal{R}')$. Jedoch ist die Berechnung einer globalen Diagnose für große Ontologien nicht effizient [20]. Die Identifikation potentiell konfliktbehafteter Korrespondenzen basiert zum Teil auf folgendem Muster:

$$A_1, B_1 \in O_1 \text{ und } A_2, B_2 \in O_2 : A_1 \sqsubseteq B_1 \wedge A_2 \sqsubseteq \neg B_2 \wedge A_1 \equiv A_2 \wedge B_1 \equiv B_2$$

Die Korrespondenzen $\{(A_1, A_2), (B_1, B_2)\}$ stellen eine Konfliktmenge dar, da A_2 ein Subkonzept von B_2 sein müsste, diese jedoch nach der Definition disjunkt sind. Im Folgenden wird das Verfahren für die unvollständige Berechnung der lokalen Diagnose nach [20] näher erläutert.

Das Verfahren ist im Algorithmus 2 dargestellt. Die Eingabe sind die Ontologien O_1 und O_2 und das Mapping $\mathcal{M}$. Die Ausgabe sind das reparierte Mapping $\mathcal{M}'$ und die Reparaturmenge $\mathcal{R}^{\approx}$. Das reparierte Mapping $\mathcal{M}'$ und $\mathcal{R}^{\approx}$ werden leer initialisiert. Da die Subsumption für die Identifikation der Konfliktmengen der Korrespondenzen mittels des Musters essentiell ist, werden die Ontologien O_1 und O_2 mit einem vollständigen Reasoner klassifiziert (Zeile 3). Anschließend werden alle Konfliktmengen mithilfe der Methode `computeConflPairs` ermittelt (Zeile 4). Diese Berechnung umfasst den größten Anteil an der Rechenzeit des Verfahrens. Anschließend werden sukzessive das modifizierte Mapping und die Reparaturmenge generiert, indem über alle Korrespondenzen in absteigender Reihenfolge bzgl. der Konfidenz der Menge $\mathcal{M}$ iteriert wird sowie über die Korrespondenzen der Menge $\mathcal{M}'$. Die Sortierung wird durchgeführt, da angenommen wird das Korrespondenzen mit einer hohen Konfidenz korrekt sind. Wenn ein Paar (m, m') mit $m \in \mathcal{M}$ und $m' \in \mathcal{M}'$ in der Konfliktmenge enthalten ist, ist eines der Korrespondenzen inkorrekt (Zeile 9-11). Da $\mathcal{M}'$, das reparierte Mapping repräsentiert, wird die aktuelle Korrespondenz m als inkorrekt angesehen und zu $\mathcal{R}^{\approx}$ hinzugefügt. Wenn das Paar nicht in der Konfliktmenge enthalten ist, wird die Korrespondenz zu $\mathcal{M}'$ hinzugefügt.

Algorithmus 2: Alcomo Verfahren für die Berechnung einer unvollständigen, lokalen Diagnose

```
   input : O1, O2, M
   output: R≈, M'
 1 M' ← ∅
 2 R≈ ← ∅
 3 classify(O1, O2)
 4 C ←computeConflPairs(O1, O2, M)
 5 orderByConfidence(M)
 6 foreach m ∈ M do
 7     coh ← true
 8     foreach m' ∈ M' do
 9         if (m, m') ∈ C then
10             coh ← false
11             break
12     if ¬coh then
13         R≈ ← R≈ ∪ {m}
14     else
15         M' ← M' ∪ {m}
16 return < R≈, M' >
```

3.4 Modulbasierte Repair-Methode mit Verwendung von Heuristiken

Dieses Verfahren [21] ist Teil des AgreementMakerLight Match- Verfahrens [12] und verwendet für die Berechnung der Reparaturmenge eine Modularisierung der Ontologien. Aufgrund der Zugehörigkeit zum AgreementMakerLight Verfahren wird die Methode im Folgenden als AML bezeichnet. Es werden ausschließlich Disjunktheitsaxiome und Subsumptionen betrachtet. Durch die Modularisierung wird die Menge der Entitäten reduziert, so dass die Rechenkomplexität reduziert wird. Für die Berechnung der Reparaturmenge müssen alle Teilmengen des Mappings bestimmt werden, die einen Widerspruch verursachen. Die Reparaturmenge wird heuristisch bestimmt, indem die häufigsten Korrespondenzen innerhalb der Konfliktmengen zu der Reparaturmenge hinzugefügt werden bis die Menge der Konfliktmengen leer ist.

Die Modularisierung $O'_1 \cup O'_2 \cup \mathcal{M}$ umfasst alle Konzepte und Subsumptionen, die folgende Eigenschaften erfüllen:

1. Alle Konzepte A und B, die disjunkt zueinander sind.
2. Alle Konzepte A, die in einer Korrespondenz des Mappings enthalten sind.
3. Das Konzept A, wenn es direkte disjunkte Superkonzepte C und D besitzt und kein Subkonzept B subsumiert, das ebenfalls von zwei disjunkten Konzepten C' und D' subsumiert wird.
4. Die is_a Hierarchie der originalen Ontologien wird für die Konzepte des Moduls übernommen.

Die Menge der Konzepte, die Bedingung 3. erfüllen, wird als *Checkset* bezeichnet. Die Berechnung der Reparaturmenge ist in drei Schritte untergliedert:(1) die Identifikation der Konfliktmengen,(2) die Filterung der Mengen und (3) die Generierung eines reparierten Mappings sowie der Reparaturmenge. Die Bestimmung der Konfliktmengen erfolgt durch eine Tiefensuche für jedes Konzept der *Checkset* Menge entlang der is_a Struktur der Ontologie von Subkonzept zu Superkonzept. Eine Konfliktmenge beinhaltet alle Korrespondenzen, die bei der Tiefensuche durchlaufen werden.

Die Eingabe der Methode für die Berechnung der Reparaturmenge sind die Menge C der ermittelten Konfliktmengen und das ursprüngliche Alignment $\mathcal{M}$. Die Ausgabe ist das reparierte Alignment. Die Methode terminiert, wenn keine Konfliktmenge existiert. Eine Konfliktmenge wird entfernt, wenn eine Korrespondenz aus der Menge entfernt wird.
Im Schritt 2. wird eine Filterung für alle Konfliktmengen durchgeführt, dabei werden Korrespondenzen mit einer höheren Konfidenz für das reparierte Alignment präferiert. Die Intention der Filterung ist die Reduktion der Konfliktmengen. Bei der Filterung werden für eine Konfliktmenge die Korrespondenzen absteigend bzgl. der Konfidenz sortiert. Es wird die Korrespondenz mit der niedrigsten Konfidenz entfernt, wenn innerhalb eines gegebenen *Konfidenzintervalls* ϵ keine weitere Korrespondenz mit dem entsprechenden Konfidenzintervall existiert. Für die kleinste Konfidenz c_1 und zweitkleinste Konfidenz c_2 muss die Bedingung gelten: $c_1 + \epsilon < c_2 - \epsilon$.

Im Schritt 3. wird das reparierte Alignment mithilfe der gefilterten Konfliktmengen bestimmt. Die Berechnung des reparierten Mappings ist im Algorithmus 3 dargestellt. Die optimale Lösung des Problems ist die Berechnung einer minimalen Menge von Korrespondenzen, die alle Konfliktmengen schneidet. Jedoch ist diese Berechnung für große Matchprobleme nicht effizient. Um eine approximierte Lösung zu generieren existieren zwei Varianten für die Berechnung der Reparaturmenge: mit Clustering und ohne Clustering. Beim Clustering werden die Konfliktmengen zu einem Cluster zusammengefasst, deren Schnittmenge nicht leer ist. Durch das Clustering können die Konfliktmengen innerhalb eines Clusters unabhängig von den Konfliktmengen der anderen Cluster eliminiert werden. Die Ermittlung der Reparaturmenge wird iterativ durchgeführt bis die Menge $\mathcal{P}_C$ leer ist. Basierend auf dem *isClustering* Parameter wird ein Clustering der Konfliktmengen durchgeführt (Zeile 3-6). Beim Clustering umfasst die Menge $\mathcal{P}_C$ die Menge von Clustern, die Konfliktmengen beinhalten. Wenn kein Clustering verwendet wird, umfasst die Menge $\mathcal{P}_C$ eine Menge, die sämtliche Konfliktmengen umfasst. In jeder Iteration (Zeile 7-17) wird eine Menge $\mathcal{S}$ von $\mathcal{P}_C$ gewählt (Zeile 8). Für diese Menge wird die Korrespondenz w mittels der Methode `getWorstMapping` ermittelt, die am häufigsten in den Konfliktmengen der Menge $\mathcal{S}$ vertreten ist (Zeile 10). Wenn mehrere Korrespondenzen gleich häufig vorkommen, wird die Korrespondenz gewählt, deren Löschung die meisten Konflikte auflöst. Die Ermittlung der gelösten Konflikte, wird durch eine begrenzte Tiefensuche realisiert, die durch den Parameter *sDepth* gesteuert wird. Die ermittelte Korrespondenz wird

Algorithmus 3: Modulbasiertes Map- Repair Verfahren mit Verwendung von Heuristiken

input : C : Liste von Konfliktmengen von Korrespondenzen, $\mathcal{M}$: zu reparierendes Mapping, ϵ: Konfidenzintervall, *sDepth*: Tiefe der Suche, *isClustering*: boolean
output: $\mathcal{M}$

1 **if** $\epsilon \geq 0$ **then**
2 $C \leftarrow$ `filterConflictSets`(C, ϵ)
3 **if** *isClustering* **then**
4 $\mathcal{P}_C \leftarrow$`clusterConflictSets`(C)
5 **else**
6 $\mathcal{P}_C \leftarrow \{C\}$
7 **while** $|\mathcal{P}_C| > 0$ **do**
8 $\mathcal{S} \leftarrow$ `getSet`$(\mathcal{P}_C)$
9 $\mathcal{P}_C \leftarrow \mathcal{P}_C \backslash \mathcal{S}$
10 $w \leftarrow$ `getWorstMapping`$(\mathcal{S}, \mathcal{M}, sDepth)$
11 $\mathcal{M} \leftarrow \mathcal{M} \backslash w$
12 $\mathcal{S} \leftarrow$`removeMapping`$(\mathcal{S}, w)$
13 **if** $|\mathcal{S}| > 0 \wedge$ *isClustering* **then**
14 $\mathcal{P}_\mathcal{S} \leftarrow$ `clusterConflictSets`$(\mathcal{S})$
15 $\mathcal{P}_C \leftarrow \mathcal{P}_C \cup \mathcal{P}_\mathcal{S}$
16 **else if** $|\mathcal{S}| > 0$ **then**
17 $\mathcal{P}_C \leftarrow \{\mathcal{S}\}$
18 **return** $\mathcal{M}$

aus dem ursprünglichen Mapping $\mathcal{M}$ entfernt. Die Konfliktmengen der Menge $\mathcal{S}$ werden bzgl. der Löschung der Korrespondenz *w* mittels der Methode `removeMapping` aktualisiert. Wenn die Menge $\mathcal{S}$ nicht leer ist und das Clustering aktiviert ist, wird die Menge $\mathcal{S}$ nach dem beschriebenen Kriterium neu geclustert (Zeile 14). Die generierten Mengen der Cluster werden zu der Menge $\mathcal{P}_C$ hinzugefügt. Wenn kein Clustering aktiviert ist, wird die Menge P_C auf die Menge $\{\mathcal{S}\}$ gesetzt, die die neue Menge von Konfliktmengen repräsentiert (Zeile 17). Wenn die Menge $\mathcal{P}_C$ leer ist, terminiert das Verfahren und es wird das reparierte Mapping $\mathcal{M}$ als Ergebnis zurück gegeben. Durch die Mengendifferenz von dem ursprünglichen Mapping und dem reparierten Mapping ist die Reparaturmenge ermittelbar.

3.5 Abgrenzung der eigenen Arbeit

Bisher existieren nur wenige Verfahren, die sich mit der Reparatur von Ontologie- Mappings befassen. Das Ziel dieser Arbeit ist die Entwicklung eines generischen Verfahrens, das in die *Match* Komponente des GOMMA Systems als Post- Processing Schritt integriert werden kann. Die Kernidee des entwickelten Verfahrens ist die effiziente Identifikation von unerfüllbaren Konzepten und inkorrekten Korrespondenzen mittels eines *Graphen*. Ähnlich zu bisherigen Verfahren ist es ebenfalls nicht vollständig, um effizient und skalierbar zu sein. Es beschränkt sich auf Subsumptionen und Disjunktheitsbeziehungen. Die Konzepte und die is_a Hierarchie sowie das Mapping bilden die Knoten bzw. Kanten des Graphen, ähnlich wie beim modulbasierten Verfahren. Jedoch benötigt das implementierte Verfahren keine Minimierung der Ontologien durch eine Modularisierung, aufgrund der effizienten Identifikation der unerfüllbaren Konzepte mittels des Dijkstra- Algorithmus. Da die is_a Hierarchie essentiell für das implementierte Verfahren ist, werden die Ontologien ebenfalls wie bei Alcomo vorab klassifiziert, um den Graphen mit weiteren Kanten anzureichern.
Im Gegensatz zu LogMap erfolgt die Bestimmung der Reparaturmenge graphbasiert und nicht durch die Überprüfung der Erfüllbarkeit einer logischen Formel. Bei der Bestimmung der Reparaturmenge wird ein ähnlicher Ansatz verwendet wie bei AML. Dieser Ansatz definiert diejenigen Korrespondenzen als inkorrekt, die eine hohes Auftreten haben bzgl. der Unerfüllbarkeit der Konzepte. Das implementierte Verfahren verwendet anstatt der Filterung der Korrespondenzen durch die dazugehörigen Konfidenzen eine Gewichtung der Anzahl des Auftretens der Korrespondenzen und der Ähnlichkeit der involvierten Konzepte. Die Ähnlichkeit der Konzepte wird durch die Konfidenz der Korrespondenz und der *strukturellen Ähnlichkeit* repräsentiert.

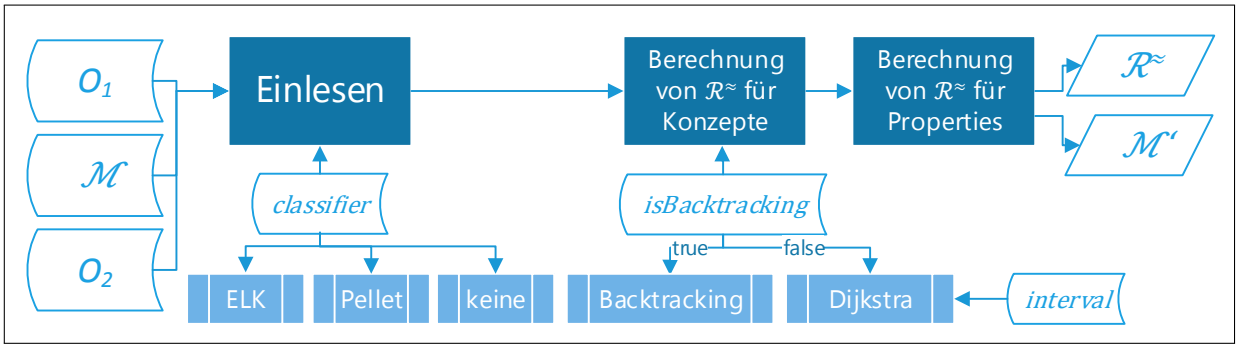
Kapitel 4

Graphbasiertes Verfahren

Das Verfahren betrachtet die Ontologie als Graph $\mathcal{G}$. Die Knoten V des Graphen werden durch die Konzepte repräsentiert und die Kanten E durch die Subsumptionen und die berechneten Korrespondenzen. Bei den Korrespondenzen werden ausschließlich Äquivalenzbeziehungen betrachtet. Jedoch ist eine Erweiterung des Verfahrens für semantisch angereicherte Mappings [22] realisierbar.

Mithilfe des Graphen ist es möglich eine Menge von inkorrekten Korrespondenzen zu ermitteln, so dass ein Großteil der Konzepte bzgl. aller Subsumptionen und Disjunktheitsaxiome der beiden Ontologien erfüllbar ist. Für die Reparatur der Korrespondenzen zwischen Properties werden die Domain- und Range betrachtet. Die Annahme ist, dass zwei Properties gleich sein können, wenn die Schnittmenge der erweiterten Domain- bzw. Ranges nicht leer ist. Die Erweiterung der Bereiche erfolgt durch die Hinzunahme der Konzepte, die eine Korrespondenz mit einem Konzept des ursprünglichen Bereichs aufweisen.
Der Ablauf des graphbasierten Verfahrens ist in der Abb. 4.1 dargestellt. Die Eingabe des Verfahrens beinhaltet die beiden Ontologien, das Mapping sowie die spezifischen Parameter *interval*, $classifier$ und *isBacktracking*. Die Parameter werden für die Reparatur des Alignments zwischen den Konzepten verwendet. Die Ausgabe des Verfahrens umfasst die Reparaturmenge $\mathcal{R}^{\approx}$ und das reparierte Mapping $\mathcal{M}'$.

Abbildung 4.1: Ablauf des graphbasierten Verfahrens. Die weiß ausgefüllten Figuren repräsentieren die Ein- und Ausgabeobjekte bzw. Parameter. Die dunkelblauen Rechtecke stellen die wesentlichen Phasen des Verfahrens dar. Die hellblauen Rechtecke symbolisieren Teilprozesse der jeweiligen Phase, deren Durchführung vom jeweiligen Parameter abhängig ist.



Das Verfahren ist in drei Phasen untergliedert. Die erste Phase umfasst das Einlesen der Ontologie. Abhängig vom $classifier$ Parameter werden die Ontologien mithilfe eines Klassifizierungsverfahren mit weiteren Subsumptionen angereichert. In der zweiten Phase wird die Reparaturmenge für die Korrespondenzen der Konzepte, basierend auf dem Ontologiegraph, berechnet. Die Identifikation der unerfüllbaren Konzepte wird durch eine Traversierung des Graphen realisiert. Die Art der Traversierung ist abhängig vom *isBacktracking* Parameter und kann ein Backtracking Ansatz oder der Dijkstra Algorithmus sein. Da der Dijkstra Algorithmus unvollständig ist bzgl. der Identifikation aller inkorrekten Korrespondenzen muss eine Neuberechnung durchgeführt werden. Der Parameter *interval* repräsentiert das Intervall zwischen den

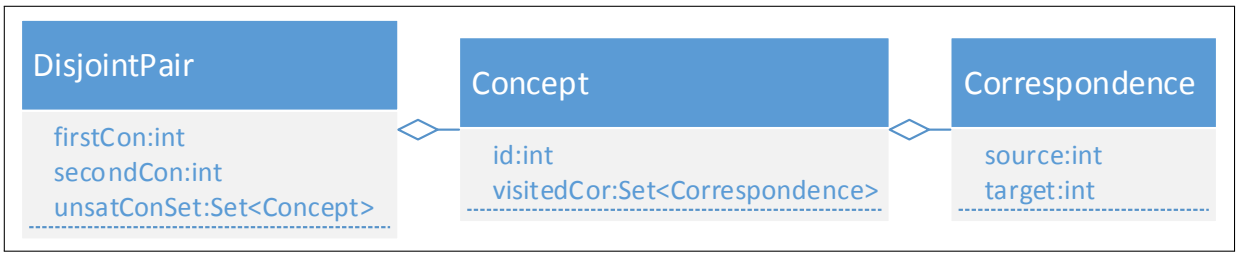
Neuberechnungen. Der letzte Schritt ermittelt inkorrekte Property- Korrespondenzen basierend auf dem reparierten Alignment. Im Folgenden werden die einzelnen Phasen näher beschrieben.

4.1 Einlesen der Ontologien

Das Verfahren benötigt als Eingabeparameter zwei Ontologien und das Alignment bzgl. der Ontologien. Wenn die Ontologien im Dateiformat vorliegen, werden sie mittels der OWL-API[1] eingelesen. Die zu extrahierenden Objekte umfassen die Konzepte, die is_a Relationen und die Disjunktheitsbeziehungen zwischen zwei Konzepten sowie die Properties mit ihren Domain und Range Bereichen. Jedes Konzept und jede Property erhält einen eindeutigen Identifier durch ein Dictionary- Encoding. Das Dictionary ist ein Mapping zwischen Identifier und IRI. Das Alignment kann entweder durch die Transformation des Alignments in die entsprechende Datenstruktur verarbeitet werden oder automatisiert ausgelesen werden, wenn es bereits im RDF Format vorliegt und den Konventionen der OWL-API entspricht. Da die *is_a* Struktur essentiell ist für die Identifikation der unerfüllbaren Konzepte, können die Ontologien mithilfe des *ELK-Reasoner* (siehe Abschnitt 2.3.2) oder durch *Pellet* klassifiziert werden.

Die extrahierten Entitäten werden zu Objekten transformiert, die der Datenstruktur in Abbildung 4.2 entsprechen. Die Klasse *Concept* repräsentiert ein Konzept, welches einen eindeutigen Identifier *id* beinhaltet. Des Weiteren enthält jedes Konzept eine Menge *visitedCor*. Diese Menge wird leer initialisiert und wird später verwendet, um die potentiellen inkorrekten Korrespondenzen bzgl. eines unerfüllbaren Konzepts zu verwalten. Die Klasse *Correspondence* stellt eine Äquivalenzbeziehung zwischen zwei Konzepten dar. Die Klasse *DisjointPair* ist die Realisierung der Disjunktheitsbeziehung von zwei Konzepten $firstCon$ und *secondCon*. Die Menge $unsatConSet$ beinhaltet unerfüllbare Konzepte, die durch diese Disjunktheitsbeziehung entstehen.

Abbildung 4.2: Datenstruktur für die Verwaltung von unerfüllbaren Konzepten bzgl. einer Disjunktheitsbeziehung mit den verantwortlichen Korrespondenzen



Die Datenstruktur $\mathcal{G}$ verwaltet die oben beschriebenen Objekte sowie die *is_a* Relationen von zwei Ontologien O_1 und O_2 in folgender Form:

[1] http://owlapi.sourceforge.net/

- *conMap*
  conMap ist eine Key-Value Map, die *Concept* Objekte verwaltet. Der Key ist der Identifier des Konzepts und der Value das entsprechende *Concept* Objekt.

- is_aEdges
  Diese Struktur ist eine Key-Value Map, die die is_a Relationen der Ontologien zwischen Konzepten verwaltet. Der Key der Map ist die *id* des Superkonzepts und der Value eine Menge von ids der Subkonzepte.

- $eqEdges$
  Diese Struktur verwaltet die Korrespondenzen in einer Key-Value Map. Der Key ist der *source* oder *target* Wert einer Korrespondenz. Der Value ist eine Menge von Korrespondenzen, die das *source* oder *target* Konzept enthalten. Bei dem Einlesen des Mappings werden die Korrespondenzen doppelt gespeichert, indem eine Korrespondenz zu der Menge der Korrespondenzen bzgl. des *source* Konzepts bzw. zu der Menge von Korrespondenzen bzgl. des *target* Konzepts abgelegt werden. Die erhöhte Speicherung ist aufgrund des Dictionary- Encoding gering, jedoch ist die Anfrage aller Korrespondenzen bzgl. eines Konzepts effizienter als eine Speicherung ohne Redundanz.

- $disjointPairs$
  Es handelt sich um eine Menge von $DisjointPair$ Objekten, die in den beiden Ontologien existieren.

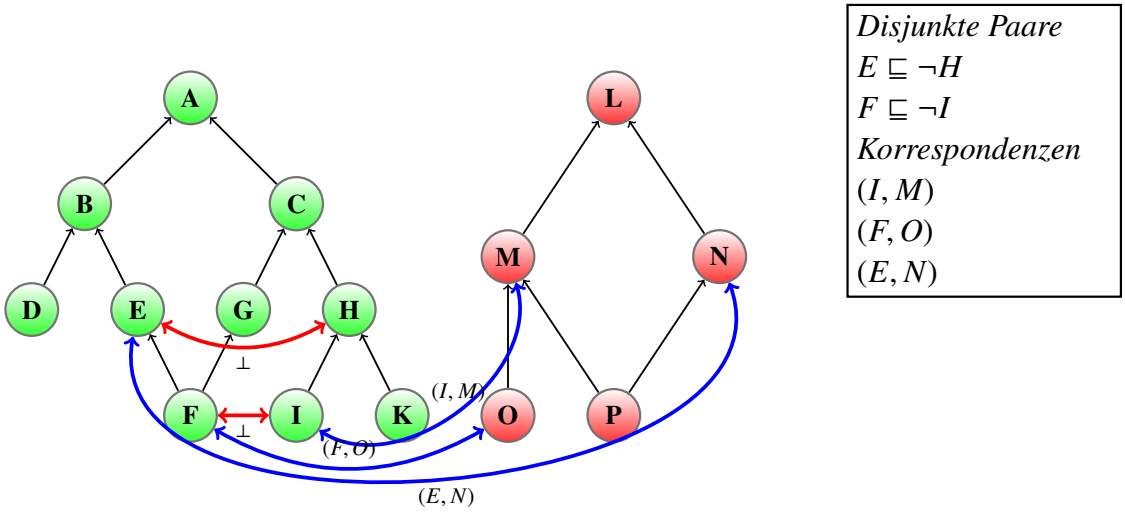
4.2 Berechnung der Reparaturmenge für Konzepte

Zur Berechnung der Reparaturmenge der Konzepte werden alle Paare von disjunkten Konzepten, is_a Relationen und Korrespondenzen betrachtet. Für die Unerfüllbarkeit von Konzepten einer Ontologie gilt die Eigenschaft:

$$\forall C \in O_1 \cup O_2 : \exists A, B \in O_1 \cup O_2 : C \sqsubseteq A \wedge C \sqsubseteq B \wedge A \sqsubseteq \neg B \rightarrow O_1 \cup O_2 \cup \mathcal{M} \models C \sqsubseteq \bot$$

Dies bedeutet, dass ein Konzept C in $O_1 \cup O_2$ unerfüllbar ist, wenn es ein Subkonzept von einem Konzept A und B ist, wobei die Konzepte A und B disjunkt sind. Zur Veranschaulichung zeigt die Abbildung 4.3 eine Beispielontologie mit einem Mapping. Die Konzepte O, P und F sind unerfüllbar. Für die Konzepte O und F gilt: $O \sqsubseteq I \wedge O \sqsubseteq F \wedge F \sqsubseteq \neg I$ bzw. $F \sqsubseteq F \wedge F \sqsubseteq I \wedge F \sqsubseteq \neg I$, so dass die Eigenschaft erfüllt ist. Für das Konzept P gilt: $P \sqsubseteq E \wedge P \sqsubseteq H \wedge E \sqsubseteq \neg H$. Aufgrund der Eigenschaft sind demzufolge alle Konzepte unerfüllbar, die von zwei disjunkten Konzepten subsumiert werden. Konzeptionell ist die Menge der unerfüllbaren Konzepte bzgl. zwei disjunkter Konzepte, die Schnittmenge der Konzepte, die von den zwei disjunkten Konzepten subsumiert werden. Unter der Annahme, dass die Ontologien ohne das Alignment kohärent sind, ist eine nicht leere Schnittmenge ausschließlich auf ein inkorrektes Alignment zurückzuführen. Der Schnitt ist z.B. für die Konzepte F und I nicht leer. Sie besitzen als gemeinsames Subkonzept das Konzept O. Aufgrund des gemeinsamen Subkonzepts besitzen die Konzepte ebenfalls eine Schnittmenge ihrer Interpretationen. Sei

Abbildung 4.3: Beispielhafte Repräsentation zweier Ontologien O_1(grün) und O_2(rot) mit einem Alignment(blau) und den Disjunktheitsbeziehungen (rot) als Graph



$O^{\mathcal{I}} = \{a, b\}$, da $O \sqsubseteq F$ und $O \sqsubseteq I$ gilt: $O^{\mathcal{I}} \subseteq F^{\mathcal{I}}$ und $O^{\mathcal{I}} \subseteq I^{\mathcal{I}}$. Daraus folgt, dass der Schnitt der Interpretationsmengen $F^{\mathcal{I}}$ und $I^{\mathcal{I}}$ nicht leer ist. Dies ist ein Widerspruch bzgl. der Disjunktheit der beiden Konzepte.

Die Menge der potentiell inkorrekten Korrespondenzen wird durch eine Traversierung von zwei Konzepten, die disjunkt sind, ermittelt. Eine Korrespondenz ist potentiell inkorrekt, wenn sie von einem Konzept des disjunkten Paares zu einem unerfüllbaren Konzept führt. Bei der Traversierung wird die *is_a* Kante von Superkonzept zu Subkonzept durchlaufen.
Es werden folgende Konzepte der disjunkten Konzepte F und I im Beispiel subsumiert. Die Menge der subsumierten Konzepte von F ist $F_{sub} = \{O, F\}$ und die subsumierte Menge von I ist $I_{sub} = \{I, M, O, P, F\}$. Die Schnittmenge ist $F_{sub} \cap I_{sub} = \{F, O\}$. Der Pfad von F nach F existiert nicht, jedoch gilt trivialerweise $F \sqsubseteq F$. Der Pfad von I nach F ist $[(I, M); (M, O); (F, O)]$. Der Pfad von F nach O ist $[(F, O)]$ und der Pfad von I nach O ist $[(I, M); (M, O)]$. Somit sind die Korrespondenzen (I, M) und (F, O) potentiell inkorrekte Korrespondenzen bei der Betrachtung des disjunkten Paares (F, I).

Um die Reparaturmenge ausgehend von der Menge der unerfüllbaren Konzepte mit den beteiligten Korrespondenzen zu ermitteln, wird ein ähnlicher Ansatz wie in [21] verfolgt. Bei diesem Ansatz wird die Häufigkeit des Auftretens einer Korrespondenz innerhalb der Konfliktmengen für die Berechnung der Reparaturmenge verwendet. Im Gegensatz dazu, bestimmt dieses Verfahren die Anzahl der Durchläufe bei der Traversierung für eine Korrespondenz, die zu einem unerfüllbaren Konzept führt. Es wird angenommen, dass eine Korrespondenz mit einer hohen Anzahl an Durchläufen inkorrekt ist. Der Algorithmus 4 stellt die Phase der Berechnung der Reparaturmenge für die Konzepte dar. Die Funktion `countConflictCor` (Zeile 1) umfasst die Traversierung des Graphen für jedes disjunkte Konzeptpaar und die Ermittlung der

Anzahl der Durchläufe für jede Korrespondenz bzgl. der unerfüllbaren Konzepte. Der Parameter *isBacktracking* ist ein boolean Parameter, der die Art der Traversierung bestimmt (siehe Abb. 4.1 "Berechnung von $\mathcal{R}^{\approx}$"). Es werden zwei Arten der Traversierung unterschieden:

- *Backtracking*(*isBacktracking = true*)
  Bei dieser Art werden alle möglichen Pfade ausgehend von einem Konzept ermittelt. Jedoch ist die Zeitkomplexität, aufgrund des Backtracking Ansatzes, exponentiell, so dass dieser Ansatz bei kleinen Ontologien zu präferieren ist.

- *Dijkstra*
  Bei diesem Ansatz werden die Korrespondenzen ermittelt, die den kürzesten Weg von einem Konzept einer Disjunktheitsbeziehung zu einem unerfüllbaren Konzept aufweisen. Prinzipiell werden die Korrespondenzen, die in der Hierarchie bzgl. eines unerfüllbaren Konzepts eines disjunkten Paares auf einer höheren Ebene stehen, für die Reparaturmenge präferiert. Somit ist berücksichtigt, dass die Erfüllbarkeit eines Konzepts abhängig ist von der Erfüllbarkeit des Superkonzepts. Bei der Verwendung dieser Variante müssen die Häufigkeiten der Korrespondenzen im Algorithmus 6 iterativ berechnet werden, da nicht alle Pfade ermittelt werden.

  Anschließend werden die Korrespondenzen basierend auf den berechneten Häufigkeiten zur Reparaturmenge durch die Methode `generateRepSet` (Zeile 2) hinzugefügt.

Algorithmus 4: Methode `computeDForCon` für die Berechnung der Reparaturmenge bzgl. der Konzepte

input : $\mathcal{G}$, *isBacktracking*, *interval*
output: $\mathcal{R}^{\approx}$

```
1 < occurrenceMap, lookupMap, confDisPairs > ← countConflictCor(G,isBacktracking)
2 R^≈ ← generateRepSet(G,occurrenceMap,confDisPairs,lookupMap,interval)
3 return R^≈
```

Der Algorithmus 5 `countConflictCor` beschreibt die Traversierung und die Bestimmung der Häufigkeiten der Korrespondenzen. Der Algorithmus generiert folgende Hilfsdatenstrukturen, die notwendig sind für die Bestimmung der Reparaturmenge:

- *occurrenceMap*
  Diese Key-Value Map verwaltet die Anzahl der Durchläufe für eine Korrespondenz, die potentiell verantwortlich ist für die Unerfüllbarkeit eines Konzepts. Der Key der Map ist ein *Correspondence* Objekt und der Value ist die Anzahl der Durchläufe bzgl. des Erreichens eines unerfüllbaren Konzepts.

- *confDisPairs*
  Diese Menge verwaltet alle *DisjointPair* Objekte, die unerfüllbare Konzepte verursachen. Diese Menge wird für die Terminierung des Algorithmus 6 verwendet.

- *lookupMap*
  Diese Map wird aufgebaut, um die *DisjointPair* Objekte zu aktualisieren. Durch das Hinzufügen der Korrespondenzen zur Reparaturmenge, werden Konzepte erfüllbar und somit wird die Menge *unsatConSet* für die betroffenen *DisjointPair* Objekte kleiner. Wenn die *unsatConSet* Menge leer ist, wird durch das entsprechende *DisjointPair* Objekt kein Konflikt verursacht und kann somit aus der Menge *confDisPairs* gelöscht werden.

Die Eingabe des Algorithmus 5 ist der Graph $\mathcal{G}$, der für die Traversierung benötigt wird und die Art der Traversierung. Die Ausgabe ist die *occurrenceMap* Datenstruktur, die im letzten Schritt bei der Generierung der Reparaturmenge verwendet wird.

Algorithmus 5: Methode `countConflictCor` für die Traversierung und Berechnung der Vorkommen aller Korrespondenzen bzgl. der Menge der unerfüllbaren Konzepte für alle Disjunktheitsbeziehungen

```
   input : G, isBacktracking
   output: occurrenceMap // Häufigkeiten der Durchläufe für jede Korrespondenz
           lookupMap // Map von Korrespondenzen und Mengen von disjunkten Paaren
           confDisPairs // Menge von disjunkten Paaren mit unerfüllbaren Konzepten
 1 confDisPairs ← ∅
 2 lookupMap ← ∅
 3 occurrenceMap ← ∅
 4 foreach disP in G. disjointPairs do
 5     subConSet1 ←traverse(G,disP.firstCon, isBacktracking)
 6     subConSet2 ←traverse(G,disP.secondCon, isBacktracking)
 7     unsatConSet ← subConSet1 ∩ subConSet2
 8     if unsatConSet ≠ ∅ then
 9         confDisPairs.add(disP)
10         foreach unsatConSet ∈ unsatConSet do
11             foreach cor ∈ unsatConSet.visitedCor do
12                 if ¬occurrenceMap.contains(cor) then
13                     occurrenceMap.put(cor,1)
14                 else
15                     occurrenceMap.put(cor,occurrenceMap.get(cor) + 1)
16                 lookupMap.put(cor,disP)
17         disP.unsatConSet ←unsatConSet
18 return <occurrenceMap,lookupMap,confDisPairs >
```

Um die Menge der potentiell inkorrekten Korrespondenzen zu ermitteln, müssen zunächst die unerfüllbaren Konzepte identifiziert werden. Diese Menge ist durch eine Traversierung des Graphen ermittelbar. Für alle disjunkten Paare erfolgt eine Traversierung ausgehend von den involvierten Konzepten der Disjunktheitsbeziehungen entlang der *is_a* Relationen und der Korrespondenzen (Zeile 4-8). Mithilfe der Funktion `traverse` wird die Menge der subsumierten Konzepte für beide Konzepte des disjunkten Paares bestimmt (Zeile 5/6). Des Weiteren werden die traversierten Korrespondenzen zu der Menge `visitedCor` für jedes unerfüllbare

Konzept hinzugefügt. Der Ablauf der Methode ist abhängig vom Parameter $isBacktracking$ und entspricht entweder der Ausführung des Dijkstra Algorithmus ($isBacktracking = false$) oder eines Backtracking Ansatzes ($isBacktracking = true$). Wenn die Schnittmenge der beiden Knotenmengen nicht leer ist (Zeile 7/8), wird das disjunkte Paar in die Menge $confDisPairs$ aufgenommen (Zeile 9). Es wird über alle Konzepte der Schnittmenge iteriert (Zeile 10-11) und jede durchlaufene Korrespondenz wird in der $occurrenceMap$ mit dem Value 1 abgelegt, falls die Korrespondenz nicht vorhanden ist (Zeile 13). Wenn eine Korrespondenz bereits enthalten ist, wird der Zähler für die Anzahl der Durchläufe um eins erhöht (Zeile 15). Des Weiteren wird die $lookupMap$ erstellt. Wenn die Korrespondenz nicht existiert, wird ein neuer Eintrag erstellt mit einer leeren Menge von $DisjointPair$ Objekten und das entsprechende $DisjointPair$ Objekt wird hinzugefügt. Falls die Korrespondenz existiert, wird das aktuelle $DisjointPair$ Objekt der bestehenden Menge hinzugefügt (Zeile 16). Um die $lookupMap$ bzgl. der gelösten Konzepte zu aktualisieren, müssen die unerfüllbaren Konzepte zu der Menge $unsatConSet$ des aktuellen disjunkten Paares hinzugefügt werden (Zeile 17).
Die Tabelle 4.1 veranschaulicht beispielhaft das Ergebnis des Algorithmus 5 inklusive der aufgebauten $lookupMap$ Datenstruktur und der $occurenceMap$ Datenstruktur, die die Anzahl der Durchläufe für eine Korrespondenz enthält. Die Menge der konfliktbehafteten disjunkten Paare ist $confDisPairs = \{(F, I); (E, H)\}$.

Tabelle 4.1: Übersicht des Resultats der Methode `countConflictCor` für das Beispielproblem

Berechnung der unerfüllbaren Konzepte und der beteiligten Korrespondenzen		
$\mathcal{G}.disjointPairs$	$unsatConSet$	$visitedCor$
$F \sqsubseteq \neg I$	F	(I,M),(F,O)
	O	(I,M),(F,O)
$E \sqsubseteq \neg H$	F	(I,M),(F,O)
	O	(I,M),(F,O)
	N	(I,M),(E,N)

$lookupMap\,(Correspondence \rightarrow \{DisjointPair\})$		$DisjointPair.unsatConSet$
Korrespondenz	disjunkte Paare	unerfüllbare Konzepte
(I, M)	(F, I)	$\{O, F\}$
	(E, H)	$\{O, F, N\}$
(F, O)	(F, I)	$\{O, F\}$
	(E, H)	$\{O, F\}$
(E, N)	(E, H)	$\{N\}$

$occurrenceMap$	
Korrespondenz	Häufigkeit
(I,M)	5
(F,O)	4
(E,N)	1

Im letzten Schritt der Methode `computeDForCon` (Algorithmus 4) wird die Reparaturmenge für die Konzepte bestimmt, indem iterativ die Korrespondenz mit der höchsten *ge-*

wichteten Anzahl (*unsatConSet*) zu der Reparaturmenge hinzugefügt wird. Der Algorithmus 6 `generateRepSet` realisiert die Generierung der Reparaturmenge. Die Eingabe des Algorithmus ist der Graph $\mathcal{G}$, die *occurenceMap*, die *confDisPairs* Menge und der *interval* Parameter. Dieser Parameter ist eine ganze Zahl und ist notwendig für die Dijkstra Traversierungsart. Der Parameter repräsentiert die Anzahl der gelöschten Korrespondenzen bis eine neue Berechnung der *occurrenceMap* erfolgen soll. Wenn $interval = -1$ ist, ist dieser Parameter die Anzahl aller unerfüllbaren Konzepte für jedes disjunkte Paar. Die Ausgabe ist die Reparaturmenge bzgl. des Mappings der Konzepte.

Algorithmus 6: Methode `generateRepSet` für die Generierung der Reparaturmenge bzgl. der Konzepte

```
   input  : G, occurrenceMap, confDisPairs, lookupMap, interval
   output : R≈
 1 R≈ ← ∅
 2 delCon ←0 // Zähler der gelöschten Korrespondenzen
 3 while confDisPairs ≠ ∅ do
 4     currentCor ← occurrenceMap.getTopOccurrenceCor ()
 5     G.eqEdges.delete(currentCor)
 6     delCon ← delCon + 1
 7     R≈ ← R≈ ∪ {currentCor }
 8     invDisPairs ← lookupMap.get(currentCor)
 9     foreach disPair ∈ invDisPairs do
10         resolvedConcepts ← ∅
11         foreach con ∈ disPair.unsatConSet do
12             if ¬isBacktracking then // Dijkstra
13                 resolvedConcepts ∪con // keine Überprüfung bzgl. Erfüllbarkeit
14             else if checkSat (disPair,con) then // Backtracking mit Überprüfung
15                 resolvedConcepts ∪con
16         updateLookupAndOccurrences
           (lookupMap,occurrenceMap,disPair,resolvedConcepts,confDisPairs)
           // Aktualisierung der Datenstrukturen bzgl. der gelösten Konzepte
17         if disPair.unsatConSet == ∅ then
18             confDisPairs.remove(disPair)
19     if ¬isBacktracking ∧ (delCon % interval ≡ 0 ∨ confDisPairs.size == 0) then
       // Überprüfung der Neuberechnungskriterien bei Dijkstra
20         < occurrenceMap, lookupMap, confDisPairs >←
           countConflictCor(G, isBacktracking)
21 return R≈
```

Die Reparaturmenge $\mathcal{R}^{\approx}$ wird leer initialisiert und die *delCon* Variable auf 0 gesetzt. Diese Variable zählt die Anzahl der bisher gelöschten Korrespondenzen und wird für die Neuberechnung bei der Dijkstra Traversierungsart verwendet. Die Berechnung der Reparaturmenge erfolgt solange bis die *confDisPairs* Menge leer ist (Zeile 3-20). Aus der *occurrenceMap* wird

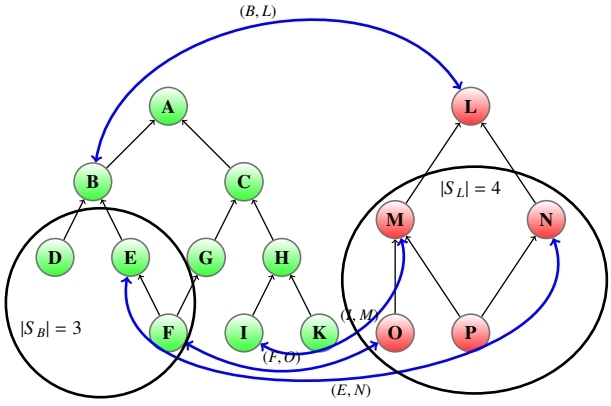
mithilfe von `getTopOccurrenceCor` die Korrespondenz mit der höchsten *gewichteten* Anzahl *unsatCount* von Durchläufen bzgl. des Erreichens eines unerfüllbaren Konzepts extrahiert und aus der Datenstruktur entfernt. Diese Korrespondenz wird ebenfalls aus der Menge der Korrespondenzen *eqEdges* des Graphen $\mathcal{G}$ entfernt (Zeile 5). Da durch das Löschen einer fehlerhaften Korrespondenz unerfüllbare Konzepte erfüllbar werden können, muss die *lookupMap* aktualisiert werden. Diesbezüglich wird die Menge der involvierten disjunkten Paare *invDisPairs* ermittelt, die in der *lookupMap* für die aktuelle Korrespondenz hinterlegt ist (Zeile 8). Für jedes disjunkte Paar werden die unerfüllbaren Konzepte in der *unsatConSet* Menge auf Erfüllbarkeit mittels der `checkSat` Methode überprüft (Zeile 14). Diese Methode berechnet die Schnittmenge der subsumierten Konzepte der beiden Konzepte von einem involvierten disjunkten Paar und überprüft ob das entsprechende Konzept in der Schnittmenge enthalten ist. Wenn das aktuelle Konzept erfüllbar ist, wird es zu der Menge der gelösten (resolved) Konzepte hinzugefügt (Zeile 15). Die Überprüfung wird ausschließlich beim Backtracking Ansatz durchgeführt, da beim Dijkstra Ansatz durch die intervallbasierte Neuberechnung die *lookupMap* und *confDisPairs* Datenstrukturen neu aufgebaut werden. Die ermittelte Menge *resolvedConcepts* wird verwendet, um die *lookupMap* und *occurrenceMap* zu aktualisieren (Zeile 16).

Bei der gewichteten Anzahl *unsatCount* der Durchläufe wird die Konfidenz einer Korrespondenz und die strukturelle Ähnlichkeit der Konzepte mit berücksichtigt. Die Anzahl *unsatCount* wird um den Faktor des Reziproken der Konfidenz $selfConf$ der Korrespondenz und um den Faktor des Reziproken der gewichteten strukturellen Ähnlichkeit $structConf$ erhöht:

$$unsatCount = unsatCount \cdot \frac{1}{selfConf} \cdot \frac{1}{w \cdot structConf + (1-w)}$$

Der Faktor der Erhöhung ist hoch, wenn sowohl die Konfidenz $selfConf$ und die strukturelle Ähnlichkeit $structConf$ niedrig ist, somit werden Korrespondenzen mit einer niedrigen Konfidenz und einer niedrigen strukturellen Ähnlichkeit für die Reparaturmenge präferiert. Mithilfe

Abbildung 4.4: Berechnung der strukturellen Ähnlichkeit für eine Korrespondenz. Die eingekreisten Konzepte repräsentieren die Menge der Konzepte des jeweiligen Subgraphen.



des Parameters w wird der Einfluss der strukturellen Ähnlichkeit gesteuert, da nicht bei allen Matchproblemen eine hohe strukturelle Ähnlichkeit auftrifft. Diese Ähnlichkeit $structConf$ einer Korrespondenz wird durch die Dice- Metrik mithilfe der Menge der Korrespondenzen spezifiziert, die die Subgraphen enthalten, und der Gesamtanzahl der Konzepte der Subgraphen. Des Weiteren sind die betrachteten Korrespondenzen nicht bei der Unerfüllbarkeit eines Konzepts involviert. Der Subgraph S_C eines Konzepts C entspricht der Menge der Subkonzepte, die durch die is_a Hierarchie der Ontologie spezifiziert ist. Die Abbildung 4.4 veranschaulicht die Berechnung der strukturellen Ähnlichkeit. Die Anzahl der Korrespondenzen der Subgraphen für die Konzepte B und L ist 2 und die Gesamtanzahl der Konzepte der Subgraphen ist 7. Daraus folgt, dass die strukturelle Ähnlichkeit $\frac{4}{7}$ ist.

Der Algorithmus 7 zeigt den konzeptionellen Ablauf der Aktualisierung. Es wird für jede Korrespondenz (Zeile 1-7), die Menge der involvierten disjunkten Paare aus der $lookupMap$ extrahiert. Für jedes disjunkte Paar (Zeile 4/5) werden die gelösten Konzepte aus der Menge der unerfüllbaren Konzepte gelöscht. Die gelöschten Konzepte müssen aufgrund der Löschung der aktuellen Korrespondenz nicht mehr für diese disjunkten Paare berücksichtigt werden, da die Unerfüllbarkeit bereits aufgelöst ist. Des Weiteren werden die Häufigkeiten für jede Korrespondenz in der $occurrenceMap$ aktualisiert, indem für jedes gelöste Konzept (Zeile 8-10) die Anzahl der durchlaufenen Korrespondenzen reduziert wird (Zeile 10).

Algorithmus 7: `updateLookupAndOccurrences` für die Aktualisierung der $lookupMap$ und Häufigkeiten der Korrespondenzen

```
   input: lookupMap , occurrenceMap, changeDp, resolvedConcepts, confDisPairs
1  foreach cor ∈lookupMap do
2      disPairSet ← lookupMap .get(cor)
3      foreach disP ∈ disPairSet do
4          if changeDp == disP then
5              disP.removeConcepts(resolvedConcepts)
6          if disP.unsatConSet == ∅ then
7              confDisPairs.remove(disP)
8  foreach con ∈ resolvedConcepts do
9      foreach cor ∈ con.visitedCor do
10         occurrenceMap.put(cor,occurrenceMap.get(cor)-1)
```

Sowohl bei der Aktualisierung als auch bei der Generierung der Reparaturmenge wird ein disjunktes Paar aus der Menge $confDisPairs$ gelöscht, wenn es keine unerfüllbaren Konzepte enthält. Bei der Traversierung mithilfe des Dijkstra Algorithmus wird eine Neuberechnung durchgeführt, wenn die Anzahl der gelöschten Korrespondenzen ein Vielfaches von $interval$ ist. Bei der Neuberechnung wird die $occurrenceMap$, $lookupMap$ und die $confDisPairs$ Menge neu berechnet. Des Weiteren wird eine Neuberechnung durchgeführt, wenn die $confDisPairs$ Menge leer ist, da die vermeintlich gelösten Konzepte ohne Überprüfung gelöscht werden (Zeile

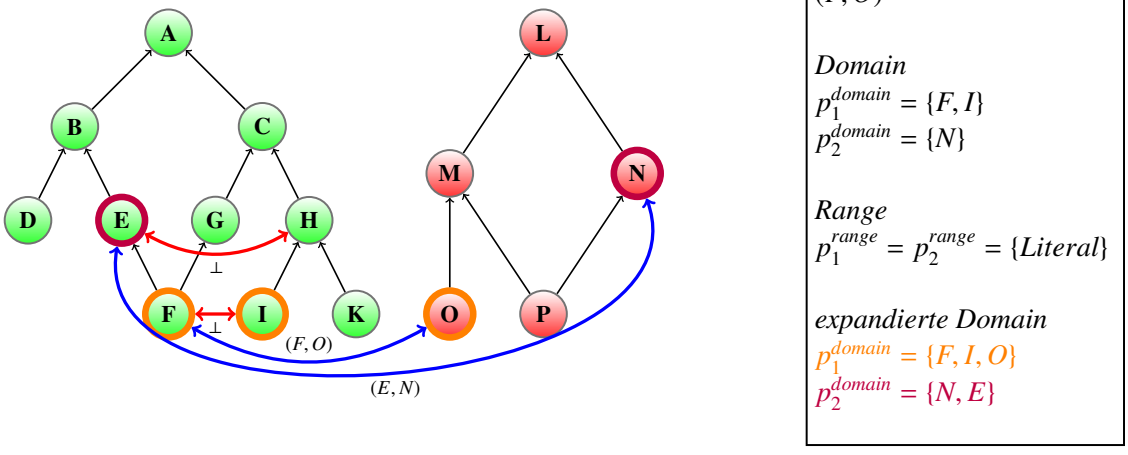
13). In der Evaluierung werden verschiedene Werte für diesen Parameter getestet. Ein kleiner Wert führt zu einer hohen Anzahl von Neuberechnungen, wohingegen ein großer Wert potentiell viele Korrespondenzen löscht ohne eine Überprüfung durchzuführen, ob unerfüllbare Konzepte existieren.

Im Beispiel wird durch den Algorithmus 6 die Korrespondenz (I, M) aus dem Graphen gelöscht. Durch das Löschen der Korrespondenz wird die Menge der unerfüllbaren Konzepte für die disjunkten Paare (F, I) und (E, H) leer. Dies wird durch die `updateLookupAndOccurrences` Methode und die Datenstruktur *lookupMap* realisiert. Da die Mengen der unerfüllbaren Konzepte für die beiden disjunkten Paare leer sind, werden beide von der Menge $confDisPairs$ entfernt. Der Algorithmus terminiert und die Reparaturmenge beinhaltet die Korrespondenz (I, M).

4.3 Berechnung der Reparaturmenge für Properties

Nach der Reparatur des Alignments bzgl. der Konzepte, erfolgt die Reparatur des Alignments für die Properties. Für die Reparatur der Properties wird angenommen, dass eine Property $p_1 \in O_1$ gleich einer Property $p_2 \in O_2$ ist, wenn die Domain bzw. Range ähnlich sind. Die Ähnlichkeit von zwei Bereichen ist in diesem Sinne eine Überlappung der expandierten Bereiche. Ein Bereich wird expandiert, indem jedes Konzept, welches eine Korrespondenz mit einem Konzept des Bereiches aufweist, der Menge hinzugefügt wird. Wenn die beiden expandierten Domains bzw. Ranges keine Schnittmenge bilden, ist nicht von einer Gleichheit der Properties auszugehen. Es wird keine Gleichheit der Mengen gefordert, da das Alignment bzlg. der Konzepte im Allgemeinen nicht vollständig ist.

Abbildung 4.5: Beispiel für die Identifikation fehlerhafter Propertykorrespondenzen



Zur Veranschaulichung wird das Beispiel mit dem reparierten Alignment fortgesetzt (siehe Abb. 4.5). Des Weiteren wird angenommen, dass eine Property $p_1 \in O_1$ existiert mit der Domain $p_1^{domain} = \{F, I\}$ und eine Property $p_2 \in O_2$ mit der Domain $p_2^{domain} = \{O\}$. Die Ranges sind bei beiden Properties Literal. Die expandierten Domains sind:$p_1^{domain\_exp} = \{F, I, O\}$ und $p_2^{domain\_exp} = \{O, E\}$. Die Schnittmenge der beiden Domains ist leer, so dass $p_1 \not\equiv p_2$.

Das Resultat der beiden Reparaturverfahren für die Konzepte und Properties ist eine Menge von inkorrekten Korrespondenzen $\mathcal{R}^{\approx}$. Um ein repariertes Mapping $\mathcal{M}'$ zu erhalten, werden alle Korrespondenzen der Reparaturmenge aus dem originalen Mapping entfernt. Im Folgenden wird das hier vorgestellte Verfahren vergleichend mit anderen Reparaturverfahren evaluiert.

Kapitel 5

Evaluierung

5.1 Datensätze

Für die Evaluierung der Verfahren werden die Ontologien des Large BioMed Track von der *Ontology Alignment Evaluation Initiative* (OAEI) verwendet. Der Large BioMed Track umfasst den National Cancer Institute Thesaurus (NCI), das Foundational Model of Anatomy (FMA) und die SNOMED CT (SNMD) Ontologie. Bei der Evaluierung werden die vollständigen Ontologien verwendet mit der Ausnahme der SNOMED CT Ontologie, für die ein großer Teil betrachtet wird. Eine Übersicht über die Anzahl der Konzepte und der Anzahl der explizit definierten Subklassenbeziehungen ist in Tabelle 5.1 dargestellt. Als Eingabe werden zudem die von GOMMA, GOMMA- reuse, YAM++ und ServOMap generierten Mappings im OAEI Contest 2012 verwendet. Die Qualität des Repair- Verfahrens wird durch die Qualität des resultierenden Mappings und die Effizienz des Verfahrens bestimmt. Des Weiteren wird die Größe der Reparaturmenge betrachtet. Für die Ermittlung des Grads der Inkohärenz wird die Anzahl der unerfüllbaren Konzepte mit dem vollständigen Reasoner HermiT berechnet. Da die vereinigte Wissensbasis aus dem NCI Thesaurus und dem großen Fragment der SNOMED CT Ontologie zu komplex für einen vollständigen Reasoner ist, wird die Anzahl der unerfüllbaren Konzepte in diesem Fall mit dem ELK- Reasoner ermittelt. Um die Korrektheit und Vollständigkeit des reparierten Mappings zu bestimmen, werden die Referenzalignments des Large BioMed Tracks verwendet. Diese wurden aus dem Unified Medical Language System (UMLS) [23] extrahiert. Diese Alignments sind ein Silberstandard, da sie einige unerfüllbare Konzepte bzgl. der vereinigten Wissensbasis aufweisen und nicht vollständig sind. Aufgrund dessen, werden für die Berechnung der Precision und des Recalls die reparierten Mappings [2] von UMLS verwendet, die von den Organisatoren des Large BioMed Tracks durch Alcomo, LogMap und einer manuellen Validierung generiert wurden. Die Evaluierung wird auf einem Laptop mit 4 Kernen à 2.2 GHz und 8GB Arbeitsspeicher durchgeführt. Ein Verfahren wird abgebrochen, wenn es innerhalb von 3 Stunden nicht terminiert.

Tabelle 5.1: Übersicht der verwendeten Ontologien mit der Anzahl der Konzepte und Subsumptionen

Ontologie	$\|C\|$	$\|is\_a\|$
FMA	78989	78985
NCI	66724	59794
SNMD	122465	105566

Im Folgenden wird das graphbasierte Verfahren bzgl. der verschiedenen Konfigurationen evaluiert, um die bestmögliche Einstellung zu identifizieren. Im zweiten Abschnitt werden die Verfahren untereinander verglichen und die Ergebnisse interpretiert.

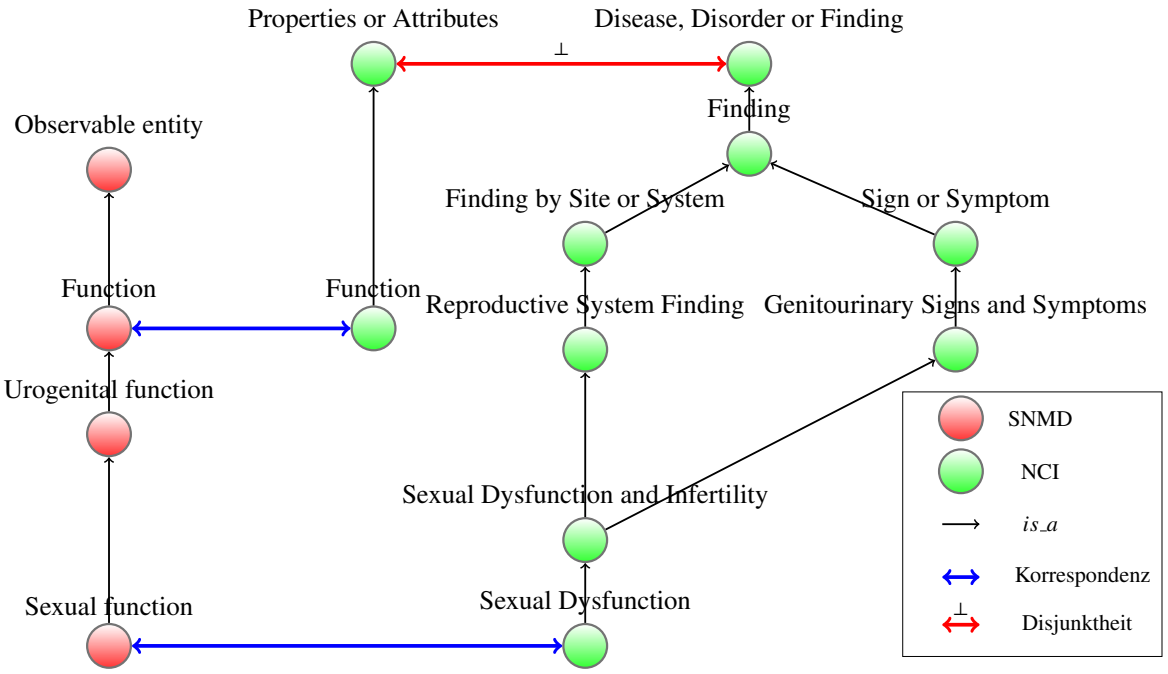
5.2 Evaluierung des graphbasierten Verfahren

Da bei der Evaluierung ausschließlich numerische Werte verwendet werden, um die Qualität darzulegen, wird zu Beginn ein kleines Realbeispiel präsentiert, das eine konkrete falsche Korrespondenz beinhaltet, die durch das Verfahren als inkorrekt identifiziert werden soll. Das Beispiel

[2] http://www.cs.ox.ac.uk/isg/projects/SEALS/oaei/2013/oaei2013_umls_reference.html

basiert auf der SNOMED CT- NCI Thesaurus Matchaufgabe. Das zu reparierende Mapping ist von GOMMA, welches beim OAEI Contest 2012 generiert wurde. Zur Veranschaulichung wird in Abbildung 5.1 eine kleine Teilmenge der Ontologien und des Mappings dargestellt. Dabei beschränkt sich die Darstellung der Konzepte auf den jeweiligen Namen. Die Korrespon-

Abbildung 5.1: Realbeispiel für die SNOMED CT- NCI Thesaurus Matchaufgabe. Der Graph mit den roten Knoten repräsentiert die SNOMED CT Ontologie und der Graph mit den grünen Knoten die NCI Thesaurus Ontologie.

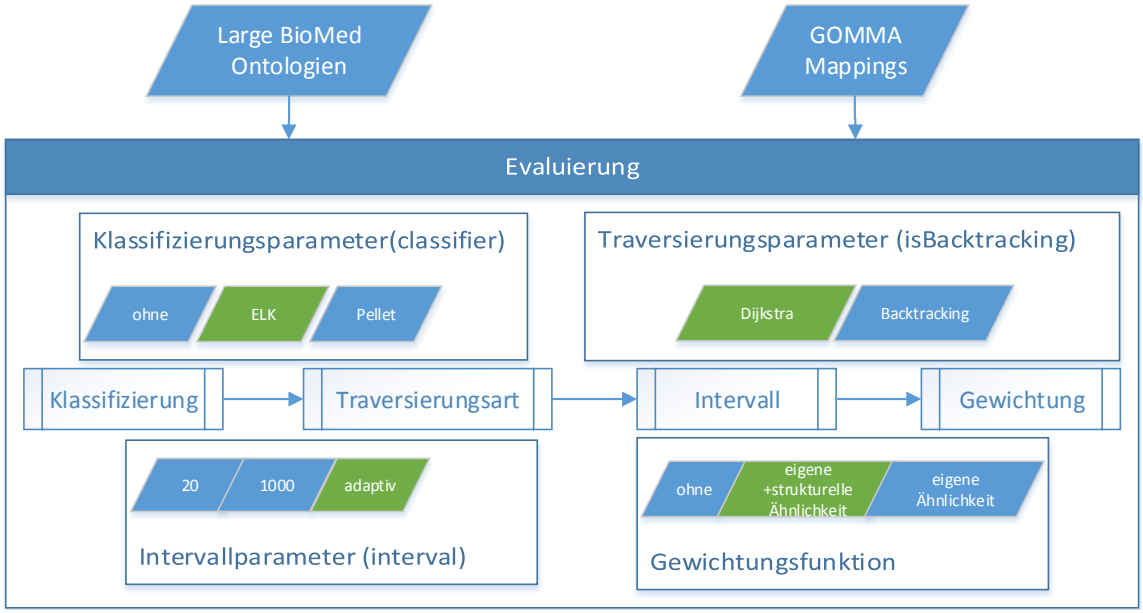


denz *(Sexual function, Sexual Dysfunction)* ist offensichtlich falsch, da es sich um gegenteilige Aspekte handelt. Eine mögliche Ursache ist der ausschließliche Einsatz von linguistischen Match- Verfahren, da die linguistische Ähnlichkeit bei der Betrachtung des Namens der beiden Konzepte sehr hoch ist. Durch die Verwendung des implementierten Verfahrens wurde diese inkorrekte Korrespondenz identifiziert. Die Identifikation erfolgte durch die Betrachtung der zwei disjunkten Konzepte *Disease, Disorder or Finding* und *Properties or Attributes*. Diese beiden Konzepte haben aufgrund der Korrespondenzen *((SNMD)Function,(NCI)Function)* und *(Sexual function,Sexual Dysfunction)* das Konzept *Sexual Dysfunction* gemeinsam. Die Korrespondenz *((SNMD)Function,(NCI)Function)* wird aufgrund der geringen Anzahl der Durchläufe und der hohen Ähnlichkeit als richtig identifiziert. Die Korrespondenz *(Sexual function,Sexual Dysfunction)* wird als falsch identifiziert.

Das graphbasierte Verfahren verwendet die Parameter $classifier$, $isBacktracking$ und $interval$. Des Weiteren wird für die Auswahl der Korrespondenzen eine Gewichtung der Anzahl der Durchläufe $unsatCount$ vorgenommen. Diesbezüglich werden verschiedene Gewichtungen verglichen. Mithilfe des $classifier$ Parameters kann der Ontologiegraph in der Einlesephase mit impliziten is_a Relationen angereichert werden. Der $isBacktracking$ Parameter definiert

die Art der Traversierung bei der Berechnung der Reparaturmenge für die Konzepte. Wenn *isBacktracking true* ist, wird bei der Traversierung ein Backtracking Ansatz verwendet. Bei dieser Variante werden alle Pfade bestimmt. Im Gegensatz dazu, ist die Traversierung mithilfe des Dijkstra Algorithmus unvollständig bzgl. der Identifikation der Pfade. Aufgrund dessen wird nach einer definierten Anzahl an gelöschten Korrespondenzen bei der Bestimmung der Reparaturmenge eine Neuberechnung durchgeführt. Die Anzahl der gelöschten Korrespondenzen wird durch den *interval* Parameter spezifiziert. Im folgenden werden verschiedene Konfigurationen bzgl. der Parameter evaluiert. Der Ablauf der Evaluierung ist in Abbildung 5.2 dargestellt. Zu

Abbildung 5.2: Ablauf der Evaluierung des graphbasierten Verfahrens. Die grün gefärbten Parameter entsprechen den Standardwerten.



Beginn wird der Einfluss der Klassifizierung gemessen. Dabei wird die Dijkstra Variante als Traversierungsart, ein *adaptives* Intervall und eine Gewichtung durch die eigene Konfidenz einer Korrespondenz sowie die strukturelle Ähnlichkeit verwendet. Bei der Berechnung der strukturellen Ähnlichkeit ist w gleich 0.6. Anschließend wird die beste Klassifizierungsvariante verwendet, um die effektivste Traversierungsart zu identifizieren. Die anderen Parameter entsprechen den Werten wie bei der Klassifizierungsevaluierung. Im 3. Schritt wird der Einfluss des Intervalls der Neuberechnungen bei der Dijkstra Traversierungsvariante betrachtet. Abschließend werden die Auswirkungen unterschiedlicher Gewichtungsfunktionen analysiert.

Die erste Konfigurationsmöglichkeit unterscheidet bei der Klassifizierung der Ontologien zwischen der Klassifizierung mittels des ELK- Reasoner sowie des Pellet- Reasoner und keiner Klassifizierung. Bei der Verwendung des Pellet- Reasoner werden alle impliziten Subsumptionen in den Reparaturprozess einbezogen. Der ELK- Reasoner identifiziert nicht alle impliziten Subsumptionen. Die Intention der Variation der Klassifizierung ist die Abwägung zwischen der zunehmenden Komplexität des Reparaturprozesses mit zunehmender Anzahl an Subsumptionen und der Identifikation der unerfüllbaren Konzepte, die auf der Traversierung des Graphen

basiert. Des Weiteren stellt die Klassifizierung einen höheren zeitlichen Aufwand dar als ohne Klassifizierung, so dass sich die Gesamtlaufzeit erhöht. Die Tabelle 5.2 stellt die Abhängigkeit des Einflusses der Klassifizierung und der Qualität bzgl. der logischen Konsistenz des reparierten Mappings dar.

Tabelle 5.2: Evaluierung der Methode ohne Klassifizierung und mit Klassifizierung durch den ELK-Reasoner sowie den Pellet- Reasoner

Daten	Klass.	|M|	|R|	Unerf.
FMA-NCI	original	5686	0	5574
	ohne	5552	**134**	12
	ELK	5542	144	**9**
	Pellet	5540	146	**9**
FMA-SNMD	original	11296	0	9918
	ohne	10874	**422**	4
	ELK	10850	446	**0**
	Pellet	10850	446	**0**
SNMD-NCI	original	27386	0	≥126406
	ohne	23298	**4088**	≥4347
	ELK	22994	4392	**≥0**
	Pellet	22844	4542	**≥0**

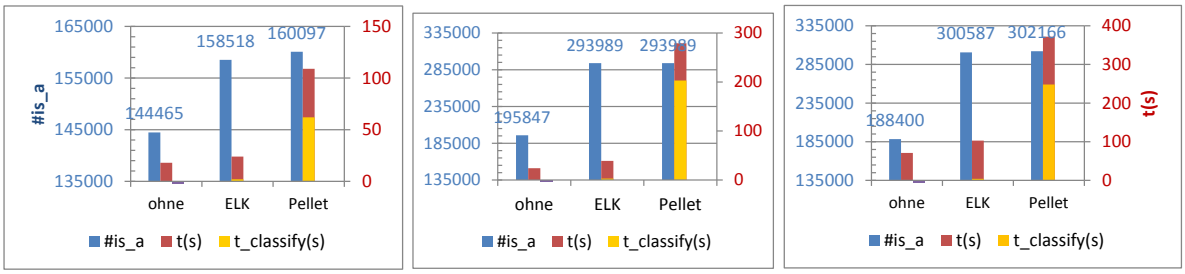
Die Spalte *Klass.* repräsentiert die Art der Klassifizierung. Die Spalten $|M|$ und $|R|$ entsprechen der Kardinalität des reparierten Mappings bzw. der Reparaturmenge. Der Grad der Inkohärenz wird durch die Anzahl der unerfüllbaren Konzepte *(Unerf.)* dargestellt. Alle Varianten verringern signifikant den Grad der Inkohärenz im Vergleich zum originalen Mapping, da die Anzahl der unerfüllbaren Konzepte bei allen Reparaturvarianten deutlich geringer ist als beim originalen Mapping. Dennoch ist die Anzahl der unerfüllbaren Konzepte nach der Reparatur ohne Klassifizierung höher als mit Klassifizierung, wie z.B. bei SNMD- NCI (≥ 4347 bzw. ≥ 0).

Die Diagramme in Abbildung 5.3 veranschaulichen die Abhängigkeit zwischen der Anzahl der Subsumptionen der beiden Ontologien und der Laufzeit des Verfahrens. Die blauen Balken repräsentieren die Anzahl der Subumptionen ($\#is\_a$) und die roten sowie die gelben Balken stellen die Gesamtlaufzeit (t(s)) bzw. die Klassifizierungszeit (t_classify(s)) dar. Bei allen Datensätzen ist der Einfluss der Klassifizierung bzgl. der Anzahl der Subsumptionen und der Laufzeit signifikant. Die Steigerung der Anzahl der Subsumptionen beträgt 14053, 98142 und 113766 Subsumptionen für die Datensätze FMA-NCI, FMA-SNMD bzw. SNMD- NCI durch die Klassifizierung mittels des ELK- Reasoners. Jedoch ist die Anzahl der Subsumptionen durch die Klassifizierung mittels des ELK- Reasoner und des Pellet- Reasoner fast gleich. Die Steigerung der Subsumptionen bzgl. des Vergleichs des ELK- Reasoner und des Pellet-Reasoner beträgt lediglich 1579, 0 und 1579.

Beide Klassifizierungsmethoden führen zu einer höheren Laufzeit als ohne Klassifizierung. Jedoch weist der Pellet- Reasoner aufgrund der Vollständigkeit hohe Laufzeiten bzgl. der Klas-

sifizierung auf, die zu einer hohen Gesamtlaufzeit des Verfahrens führen. Im Gegensatz dazu, sind die Laufzeiten des ELK- Reasoner mit $2s$, $3s$ und $4s$ sehr gering, so dass der Einfluss bzgl. der Gesamtlaufzeit klein ist. Des Weiteren verursacht eine erhöhte Anzahl der Subsumptionen eine höhere Laufzeit. Diese Beobachtung verdeutlicht insbesondere der SNMD -NCI Datensatz, der bei der ELK- Reasoner Klassifizierung im Vergleich zu der Pellet- Reasoner Klassifizierung lediglich $\approx$ 1500 Subsumptionen weniger beinhaltet. Jedoch ist das Verfahren mit der ELK Klassifizierung eine halbe Minute schneller als mit der Pellet Klassifizierung abzüglich der Klassifizierungslaufzeiten. Aufgrund der Resultate ist die Klassifizierungsart mit dem ELK-

Abbildung 5.3: Abhängigkeit der Anzahl der Subsumptionen und der Laufzeit des graphbasierten Verfahrens. Die blauen Balken repräsentieren die Anzahl der Subsumptionen. Die roten Balken und die gelben Balken stellen die Gesamtlaufzeit bzw. die Klassifizierungszeit dar.



(a) FMA-NCI **(b)** FMA-SNOMED CT **(c)** SNOMED CT-NCI

Reasoner zu präferieren, um eine gute Qualitätsverbesserung bzgl. der logischen Konsistenz des Mappings und eine annehmbare Effizienz zu gewährleisten. Wenn die Effizienz gegenüber der logischen Konsistenz präferiert wird, ist keine Klassifizierung notwendig. Des Weiteren ist keine Klassifizierung notwendig, wenn die Ontologien einfach strukturiert sind und eine geringe Menge an impliziten Subsumptionen aufweisen.

Die weitere Evaluierung verwendet für das graphbasierte Verfahren die Klassifizierung mit dem ELK- Reasoner. Die zweite Konfigurationsmöglichkeit ist die Traversierungart bei der Bestimmung der Reparaturmenge für die Konzepte. Es wird zwischen der Traversierung durch den Dijkstra Algorithmus und einem Backtracking Ansatz differenziert. Da der Dijkstra Algorithmus immer die kürzesten Wege von einem Knoten zu allen anderen Knoten identifiziert, ist eine intervallbasierte Neuberechnung notwendig. Diese Neuberechnung ist beim Backtracking Ansatz nicht notwendig, da alle Pfade ermittelt werden. Jedoch ist die Komplexität des Backtracking exponentiell und die Komplexität des Dijkstra Algorithmus ist linear, so dass zu erwarten ist, dass die Laufzeit des Dijkstra Ansatzes geringer ist als die Gesamtzeit des Backtracking Ansatzes. Die Tabelle 5.3 stellt den Vergleich der zwei Traversierungsarten dar. Dabei werden die Größe des resultierenden Mappings $|M|$ bzw. die Größe der Reparaturmenge $|R|$ und die Laufzeit $t(s)$ dargestellt. Außerdem werden die Precision (P), der Recall (R), das F- Measure (F) und die Anzahl der unerfüllbaren Konzepte (Unerf.) als Maße für die Qualität des Mappings präsentiert. Der Backtracking Ansatz generiert ausschließlich für den FMA-NCI Datensatz ein Resultat. Bei allen anderen Datensätzen terminierte der Algorithmus nicht in der vorgegebenen Zeit. Die Laufzeit des Backtracking Ansatzes ist $\approx$ 5 mal höher als beim

Tabelle 5.3: Vergleich der Traversierung mittels Backtracking Ansatz und Dijkstra Ansatz

Daten	Daten	|M|	|R|	t(s)	P	R	F	Unerf.
FMA_NCI	original	5686	0	-	0.8554	**0.8356**	**0.8454**	5574
	Dijkstra	5542	**144**	**24**	0.8667	0.8251	**0.8454**	**9**
	Backtracking	5514	172	126	**0.8687**	0.8229	0.8452	**9**
FMA-SNMD	original	11296	0	-	0.4025	**0.2642**	0.3190	9918
	Dijkstra	10850	**446**	**39**	**0.4147**	0.2614	**0.3206**	**0**
	Backtracking	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SNMD-NCI	original	27386	0	-	0.7060	**0.5311**	0.6062	≥126406
	Dijkstra	22994	**4392**	**103**	**0.8155**	0.5151	**0.6314**	**≥0**
	Backtracking	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Dijkstra Ansatz. Alle Varianten erhöhen die Precision im Vergleich zum originalen Mapping. Bei allen Methoden verringert sich der Recall. Jedoch nivelliert die Zunahme der Precision die Verringerung des Recalls bei der Dijkstra Traversierungsart, so dass das F- Measure bei allen Datensätzen höher ist als bei den originalen Mappings von GOMMA. Eine signifikante Verbesserung des F- Measures ist beim SNMD-NCI Datensatz zu beobachten, da das resultierende Mapping eine Verbesserung von $\approx$ 3% erzielt. Im Vergleich zum Backtracking Ansatz generiert die Dijkstra Variante kleinere Reparaturmengen. Eine mögliche Erklärung für die höhere Anzahl der gelöschten Korrespondenzen ist die Vollständigkeit der Traversierung. Dadurch ist es möglich, dass viele Korrespondenzen die gleiche Häufigkeit aufweisen. Bei gleicher Anzahl und gleicher Konfidenz werden die Korrespondenzen wahllos gelöscht. Jedoch ist nicht garantiert, dass die Konflikte durch die Löschung einer häufig auftretenden Korrespondenz sofort gelöst sind. Im Gegensatz dazu, ist durch die Neuberechnung und die Betrachtung der Korrespondenzen bei der Dijkstra Variante anzunehmen, dass die Häufigkeitsverteilung eine höhere Diversität aufweist als die Backtracking Variante.

Die dritte zu evaluierende Konfiguration fokussiert sich auf den *interval* Parameter, der bei der Traversierung mittels des Dijkstra Algorithmus verwendet wird. Dieser Parameter steuert die Anzahl der zu löschenden Korrespondenzen bevor eine Neuberechnung stattfindet. Wenn die Menge der konfliktbehafteten disjunkten Paare ($confDisPairs$) leer ist, wird jedoch ebenfalls eine Neuberechnung durchgeführt, um zu gewährleisten, dass alle Konflikte behoben sind. Im Folgenden wird der Parameter auf 20, 1000 und die Anzahl aller unerfüllbaren Konzepte für alle disjunkten Paare gesetzt. Die letzte Konfiguration ist abhängig von der jeweiligen Problemgröße. Wenn die Menge der unerfüllbaren Konzepte klein ist, werden bei einem großen *interval* unnötig viele Korrespondenzen gelöscht. Im Gegensatz dazu, werden bei einer großen Anzahl unerfüllbarer Konzepte und einem kleinen *interval* viele Neuberechnungen durchgeführt, um die Konflikte aufzulösen. Da die Anzahl der inkorrekten Korrespondenzen eine Abhängigkeit bzgl. der Anzahl der unerfüllbaren Konzepte aufweist, ist es sinnvoll, wenn sich die Anzahl der zu löschenden Korrespondenzen der Problemgröße anpasst. Die Tabelle 5.4 veranschaulicht die Resultate bzgl. der verschiedenen *interval* Größen. Die Resultate bestätigen die Annahme, dass

Tabelle 5.4: Evaluierung unterschiedlicher *interval* Parameter beim Dijkstra Ansatz

Daten	Intervall	|M|	|R|	t(s)	F
FMA-NCI	original	5686	0	-	0.8454
	20	5544	**142**	25	**0.8466**
	1000	5542	144	**24**	0.8454
	adaptiv	5542	144	**24**	0.8454
FMA-SNMD	original	11296	0	-	0.3190
	20	10868	**428**	46	**0.3208**
	1000	10854	442	**39**	**0.3208**
	adaptiv	10850	446	**39**	0.3206
SNMD-NCI	original	27386	0	-	0.606
	20	23050	**4336**	184	0.6312
	1000	22996	4390	**103**	0.6313
	adaptiv	22994	4392	**103**	**0.6314**

ein zu klein gewähltes *interval* zu einer schlechteren Laufzeit führt, da mehr Neuberechnungen durchgeführt werden müssen. Besonders deutlich wird dies bei der Betrachtung der Laufzeiten für den SNMD-NCI Datensatz, wo die Laufzeit für das $interval = 20$ mehr als eine Minute langsamer ist als bei den *interval* Parametern 1000 und der adaptiven Variante. Die Laufzeiten der letzteren Parameterwerte weisen eine identische Laufzeit auf. Die Ursache ist die ähnliche Anzahl an Neuberechnungen. Bei dem $interval = 1000$ wird permanent eine hohe Anzahl an Korrespondenzen gelöscht bis alle Konflikte gelöst sind. Die adaptive Variante löscht zu Beginn bei einer hohen Konfliktanzahl mehr Korrespondenzen als bei dem konstanten Intervall. Jedoch nimmt die Anzahl der zu löschenden Korrespondenzen mit sinkender Anzahl der Konflikte ab. Die Qualität der resultierenden Mappings ist gleich oder marginal unterschiedlich, wobei keine Variante mehrheitlich besser ist als die anderen Verfahren. Aufgrund der besseren Laufzeiten und der Adaptionsfähigkeit bzgl. der Problemgröße wird die adaptive Variante für den Parameter *interval* gewählt.

Abschließend werden verschiedene Gewichtungen der $unsatCount$ Variable evaluiert. Diese Variable wird für die Auswahl einer Korrespondenz bei der Generierung der Reparaturmenge verwendet. Die standardmäßige Gewichtung betrachtet die Anzahl der Durchläufe einer Korrespondenz ($unsatCount$) bzgl. des Erreichens eines unerfüllbaren Konzepts, die Konfidenz der Korrespondenz $selfConf$ und die strukturelle Ähnlichkeit ($strucConf$). Diese Gewichtung entspricht folgender Gleichung:

$$unsatCount = unsatCount \cdot \frac{1}{selfConf} \cdot \frac{1}{w \cdot structConf + (1-w)}$$

Des Weiteren werden im folgenden eine Gewichtung ohne strukturelle Ähnlichkeit und ohne Gewichtung bzgl. jeglicher Ähnlichkeit betrachtet. Die Tabelle 5.5 werden die verschiedenen Gewichtung bzgl. der resultierenden Mappingqualität mittels Precision (P), Recall (R)

und F- Measure (F) verglichen. Des Weiteren wird die Größe der Reparaturmenge ($|R|$) dargestellt. Die Qualität des Mappings ist ohne Gewichtung schlechter als mit Gewichtung. Wenn

Tabelle 5.5: Vergleich der Gewichtungsfunktionen der *unsatCount* Variable

Daten	Gewichtung	|M|	|R|	P	R	F
FMA-NCI	original	5686	0	0.8554	**0.8356**	**0.8454**
	ohne	5540	146	0.8655	0.8237	0.8441
	selfConf	5542	**144**	0.8656	0.8241	0.8443
	selfConf+structConf	5542	**144**	**0.8667**	0.8251	**0.8454**
FMA-SNMD	original	11296	0	0.4025	**0.2642**	0.3190
	ohne	10852	**444**	0.4131	0.2605	0.3195
	selfConf	10848	448	0.4145	0.2613	0.3205
	selfConf+structConf	10850	446	**0.4147**	0.2614	**0.3206**
SNMD-NCI	original	27386	0	0.7060	**0.5311**	0.6062
	ohne	22982	4404	0.8136	0.5136	0.6297
	selfConf	22986	4400	0.8149	0.5145	0.6308
	selfConf+structConf	22994	**4392**	**0.8155**	0.5151	**0.6314**

unsatCount nicht gewichtet wird, werden Korrespondenzen bei gleicher Anzahl der Durchläufe gleich behandelt, so dass Korrespondenzen mit hoher Konfidenz gelöscht werden können. Durch die Gewichtung werden Korrespondenzen mit hoher Konfidenz für das resultierende Mapping präferiert gegenüber Korrespondenzen mit niedriger Konfidenz. Aufgrund dessen, ist die Precision und der Recall bei allen Gewichtungen höher als ohne Gewichtung. Des Weiteren ist die Verbesserung der Qualität des Mappings bei der Berücksichtigung der strukturellen Ähnlichkeit höher als bei der ausschließlichen Betrachtung der Konfidenz der Korrespondenz.

5.3 Vergleich der Verfahren

Es werden im Folgenden die Reparaturverfahren Alcomo, LogMap, das modulbasierte Verfahren (AML) und das graphbasierte Verfahren verglichen. Beim Vergleich der Verfahren wird für das graphbasierte Verfahren folgende Konfiguration gewählt:

- Klassifizierung: ELK
- Traversierungsart: Dijkstra
- *interval*: adaptiv
- Gewichtung: $selfConf + structConf$

Die Konfiguration von Alcomo entspricht folgenden Einstellungen:

- lokale vs. globale Diagnose: lokal
- unvollständig vs. vollständig: unvollständig

- Klassifizierung: HermiT

Beim AML Reparaturverfahren werden die Standardeinstellungen verwendet, dass heißt es wird keine Filterung verwendet, der Tiefensuchenparameter ist 0 und die Konfliktmengen werden nicht geclustert.

Die Tabelle 5.6 vergleicht die Verfahren Alcomo, LogMap, AML und das graphbasierte Verfahren für den Datensatz FMA-NCI. LogMap und AML generieren für alle Mappings Alignments mit gleichen F-Measure oder höheren F-Measure. Das graphbasierte Verfahren generiert für das YAM++ und ServOMap Mapping Alignments mit den besten F-Measures. LogMap generiert für alle Match- Verfahren Alignments mit der durchschnittlich geringsten Anzahl von unerfüllbaren Konzepten mit 9 Konzepten, wobei das graphbasierte Verfahren mit durchschnittlich 9.17 eine ähnliche Anzahl generiert. AML generiert die kleinsten Reparaturmengen. Jedoch ist die Anzahl der unerfüllbaren Konzepte verglichen zu LogMap und dem graphbasierten Verfahren höher. Das graphbasierte ist signifikant schneller als die anderen Verfahren mit einer durchschnittlichen Laufzeit von 27 Sekunden. LogMap, AML und Alcomo sind ≈ 4,≈ 5 bzw. ≈ 11 mal langsamer als das graphbasierte Verfahren.

Tabelle 5.6: Vergleich der Ergebnisse der Reparatur des GOMMA, GOMMA- reuse, YAM++ und ServOMap Alignments für FMA-NCI

Mapping	Verfahren	|M|	|R|	t(s)	F	Unerf.
GOMMA	original	5686	0	-	0.8454	5574
	Alcomo	5546	140	185	0.8477	15
	LogMap	5535	151	118	**0.8484**	**9**
	AML	5564	**122**	156	0.8464	10
	Graphbasiert	5542	144	**24**	0.8454	**9**
GOMMA-reuse	original	6330	0	-	0.8426	12939
	Alcomo	6124	206	675	0.8455	29
	LogMap	6113	217	111	**0.8473**	**9**
	AML	6152	**178**	143	0.8447	10
	Graphbasiert	6118	212	**26**	0.8433	**9**
YAM++	original	5476	0	-	0.8700	50550
	Alcomo	5344	132	618	0.8724	10
	LogMap	5534	142	125	0.8724	**9**
	AML	5366	110	152	0.8723	13
	Graphbasiert	5354	122	**33**	**0.8728**	10
ServOMap	original	4932	754	-	0.8275	48743
	Alcomo	4764	168	596	0.8248	**9**
	LogMap	4793	139	141	0.8278	**9**
	AML	4820	**112**	146	0.8272	**9**
	Graphbasiert	4812	120	**31**	**0.8284**	**9**

Die Tabelle 5.7 vergleicht die Verfahren für den Datensatz FMA-SNMD. AML terminiert bei der Reparatur des GOMMA- reuse Mappings nicht in der vorgegebenen Zeit. Alle Verfahren

Tabelle 5.7: Vergleich der Ergebnisse der Reparatur des GOMMA, GOMMA- reuse, YAM++ und ServOMap Alignments für FMA-SNMD

Mapping	Vefahren	|M|	|R|	t(s)	F	Unerf
GOMMA	original	11296	0	-	0.3190	9918
	Alcomo	10716	580	2104	0.3182	**0**
	LogMap	10744	552	264	0.3178	**0**
	AML	10882	**414**	423	0.3205	24
	Graphbasiert	10850	446	**39**	**0.3206**	**0**
GOMMA-reuse	original	25372	0	-	**0.6928**	84612
	Alcomo	23036	2336	2045	0.6746	925
	LogMap	22355	3017	290	0.6673	1
	AML	N/A	N/A	N/A	N/A	N/A
	Graphbasiert	23292	**2080**	**139**	0.6816	**0**
YAM++	original	14028	0	-	**0.7611**	75329
	Alcomo	12816	1212	2136	0.7500	**0**
	LogMap	12831	1197	337	0.7502	**0**
	AML	12984	**1044**	425	0.7551	**0**
	Graphbasiert	12930	1098	**53**	0.7578	**0**
ServOMap	original	12566	0	-	**0.7647**	171700
	Alcomo	11422	1144	2123	0.7375	**0**
	LogMap	11231	1335	275	0.7179	**0**
	AML	11728	**838**	440	0.7551	51
	Graphbasiert	11612	954	**48**	0.7540	**0**

reduzieren die Anzahl der unerfüllbaren Konzepte, wobei das graphbasierte Verfahren für alle Mappings ein kohärentes Alignment generiert. Jedoch wird das F- Measure durch alle Reparaturverfahren reduziert, außer bei der Reparatur des GOMMA Mappings durch das graphbasierte Verfahren und AML. Die Reduzierung des F- Measures ist beim graphbasierten Verfahren und bei AML durchschnittlich gering mit einer Reduzierung von 0.0059 bzw. 0.0046. Die geringe Reduzierung des F-Measures resultiert aus der geringen Größe der generierten Reparaturmengen durch AML und dem graphbasierten Verfahren. Dennoch nivelliert die Erhöhung der Precision nicht die Senkung des Recalls, so dass das F- Measure durchschnittlich geringer ist als bei den originalen Mappings. Das graphbasierte Verfahren weist die geringsten Laufzeiten auf mit durchschnittlich 45.67 Sekunden.

Die Tabelle 5.8 vergleicht abschließend die Ergebnisse der Reparatur der Mappings für den SNMD-NCI Datensatz. AML terminiert nicht in der vorgegebenen Zeit für das GOMMA- reuse Mapping. Das graphbasierte Verfahren ist bzgl. der Kohärenz besser als die anderen Verfahren. LogMap, AML und das graphbasierte Verfahren generieren ähnliche Mappings bzgl. der Vollständigkeit und der Korrektheit mit einem durchschnittlichen Zuwachs von 0.0244, 0.0242 bzw. 0.0241. Das graphbasierte Verfahren erreicht bei der Reparatur aller betrachteten Mappings das 2. beste Ergebnis bzgl. F- Measure und verursacht die geringsten Laufzeiten.

Tabelle 5.8: Vergleich der Ergebnisse der Reparatur des GOMMA, GOMMA- reuse, YAM++ und ServOMap Alignments für SNMD-NCI

Mapping	Verfahren	|M|	|R|	t(s)	F	Unerf.
GOMMA	original	27386	0	-	0.6062	≥126406
	Alcomo	22794	4592	3252	0.6270	≥1155
	LogMap	22631	4755	344	0.6306	**≥0**
	AML	22916	4470	458	**0.6333**	≥10
	Graphbasiert	22994	**4392**	**103**	0.6314	**≥0**
GOMMA-reuse	original	34090	0	-	0.6345	≥159945
	Alcomo	28686	**5404**	4904	0.6578	≥12985
	LogMap	27785	6305	390	**0.6651**	**≥0**
	AML	N/A	N/A	N/A	N/A	N/A
	Graphbasiert	28660	5430	**315**	0.6615	**≥0**
YAM++	original	28206	0	-	0.6844	≥126353
	Alcomo	24784	3422	2635	0.7030	**≥0**
	LogMap	25014	3192	395	0.7084	≥13
	AML	25156	**3050**	450	**0.7099**	≥8
	Graphbasiert	25114	3092	**92**	0.7098	**≥0**
ServOMap	original	24924	0	-	0.6682	≥158764
	Alcomo	21936	2988	2616	0.6787	≥20
	LogMap	22193	2731	334	0.6868	≥112
	AML	22310	**2614**	701	**0.6882**	≥43
	Graphbasiert	22252	2672	**125**	0.6868	**≥8**

5.4 Fazit

Zur Übersicht sind die durchschnittlichen Ergebnisse bzgl. aller Mappings für jeden Datensatz und jedes Verfahren in der Tabelle 5.9 zusammengefasst. Resümierend ist festzuhalten, dass alle Verfahren die Anzahl der unerfüllbaren Konzepte reduzieren. LogMap und das graphbasierte Verfahren generieren bessere Ergebnisse bzgl. der durchschnittlichen Anzahl der unerfüllbaren Konzepte für die einzelnen Datensätze als Alcomo und AML. Das graphbasierte Verfahren erzeugt insbesondere für die Datensätze FMA-SNMD und SNMD-NCI die geringste durchschnittliche Anzahl von unerfüllbaren Konzepten. Die Qualität bzgl. des durchschnittlichen F- Measures des Mappings wird in der Regel von allen Verfahren erhöht außer beim FMA-SNMD Datensatz. Bei diesem Datensatz wird das F- Measure reduziert, wobei das graphbasierte Verfahren und AML eine geringe Reduzierung verursachen. Die Reparatur mit LogMap führt bei FMA-NCI und SNMD-NCI Mappings zur größten durchschnittlichen Erhöhung des F-Measures mit 0.0026 bzw. 0.0244, wobei AML und das graphbasierte Verfahren bei SNMD-NCI mit 0.0243 bzw. 0.0241 ähnliche Zuwächse des durchschnittlichen F-Measures erreichen. Da die Verbesserung des F-Measures vom Recall abhängig ist, ist bei der Reparatur eine geringe Anzahl zu löschender Korrespondenzen wünschenswert, was für alle Verfahren zutrifft. Die Größe der Reparaturmenge ist im Bereich von 2% (AML,Graphbasiert) bis 14% (LogMap).

Alle Verfahren benötigen für die Berechnung der Reparaturmenge eine annehmbare Zeit, wobei das graphbasierte Verfahren signifikant besser ist bzgl. der Laufzeit als die anderen Verfahren. Im Durchschnitt ist das graphbasierte Verfahren mindestens $\approx$ 17 mal schneller als Alcomo, mindestens $\approx$ 5 mal schneller als AML und mindestens $\approx$ 2 mal schneller als LogMap. Abschließend ist anzumerken, dass es sich bei der Evaluierung der Qualität der reparierten Mappings, um einen reparierten Silberstandard mithilfe von LogMap und Alcomo handelt. Dennoch wurden diese Alignments gewählt, da sie am logisch konsistenten sind und bisher keine anderen Referenzalignments existieren. Es ist zu erwarten, dass die beiden Reparaturverfahren bessere Ergebnisse bzgl. der Precision und somit des F-Measures generieren. Generell generieren LogMap und Alcomo größere Reparaturmengen und somit kleinere Mappings als AML und das graphbasierte Verfahren. Da das graphbasierte Verfahren und AML weniger Korrespondenzen entfernen, ist es möglich das die Korrespondenz nicht in den reparierten Mappings enthalten sind, die durch LogMap und Alcomo repariert wurden, da sie als inkorrekt identifiziert wurden.

Tabelle 5.9: Übersicht der Ergebnisse für alle Datensätze, Verfahren und Mappings. Die grau gefärbten Zeilen beinhalten keine vollständigen Durchschnittswerte, da das Verfahren nicht bei allen Datensätzen terminiert.

Daten	Verfahren	ø Δ(F)	ø t(s)	ø |R|	ø Unerf.
FMA-NCI	original	0	-	0	34955.67
	Alcomo	0.0012	518.50	161.50	15.75
	LogMap	**0.0026**	123.75	162.25	**9.00**
	AML	0.0013	149.25	**130.50**	10.50
	Graphbasiert	0.0007	**28.50**	149.50	9.17
FMA-SNMD	original	0	-	0	85649.00
	Alcomo	-0.0143	2102.00	1318.00	231.25
	LogMap	-0.0211	291.50	1525.25	0.25
	AML	-0.0046	436.67	**763.33**	25.00
	Graphbasiert	**-0.0059**	**69.75**	1144.50	**0.00**
SNMD-NCI	original	0	-	0	142867.00
	Alcomo	0.0183	3351.75	4101.50	3540.00
	LogMap	**0.0244**	365.75	4245.75	31.25
	AML	0.0242	538	**3386**	20.33
	Graphbasiert	0.0241	**158.75**	3896.50	**2.00**

Kapitel 6

Zusammenfassung

Eine Vielzahl der existierenden Matchverfahren generieren Mappings zwischen zwei Ontologien, die logisch inkorrekte Korrespondenzen enthalten. Aufgrund dessen befasst sich diese Masterarbeit mit der Konzeption und der Realisierung eines Verfahrens, welches ein Mapping modifiziert, so dass im besten Fall ein logisch korrektes Mapping erzeugt wird. Diesbezüglich wurden im ersten Kapitel die Grundlagen zu diesem Thema erörtert. Eine komplexe Ontologie ist eine Art der Wissensrepräsentation, die dem Formalismus der Beschreibungslogik unterliegt. Es wurden die basalen Bestandteile der Logik spezifiziert sowie deren Zusammenhänge. Des Weiteren wurden die verschiedenen Konstrukte der Beschreibungslogik $\mathcal{SHOIN}(D)$ vorgestellt, die der Web Ontology Language DL (OWL DL) entspricht. Anschließend wurden Reasoning- Verfahren präsentiert, die diesen Formalismus verwenden, um implizites Wissen der Ontologie zu ermitteln. Die Eingabe eines Reparaturverfahrens ist ein automatisch generiertes Ontologie- Mapping. Es wurden die verschiedenen Arten der Ontologie- Matching Verfahren vorgestellt sowie verschiedene Techniken für die Optimierung von Matchverfahren. Abschließend wurden exemplarisch die Match- Systeme GOMMA mit der Match Komponente, YAM++ und ServOMap erläutert.

Vor der Konzeption und der Realisierung eines Verfahrens setzt sich diese Arbeit vorab mit den bereits existierenden Verfahren auf diesem Forschungsgebiet auseinander. Diese Verfahren sind die Valdierungskomponente des LogMap Match- Systems [17], Alcomo [19] und ein modulbasiertes Verfahren (AML) [21]. Anschließend wurden die Konzeption und die Realisierung des im Rahmen dieser Arbeit entwickelten Verfahrens erläutert. Das graphbasierte Verfahren ist in drei Phasen untergliedert: die Einlesephase, die Berechnung der fehlerhaften Korrespondenzen bzgl. der Konzepte und die Ermittlung der inkorrekten Korrespondenzen bzgl. der Properties. Für die Identifikation dieser Korrespondenzen werden die Ontologien und das Mapping in eine Graphdatenstruktur transformiert. Der Großteil des Verfahrens konzentriert sich auf der Identifikation der fehlerhaften Korrespondenzen bzgl. der Konzepte. Die Kernidee des Verfahrens ist die Betrachtung der Paare der disjunkten Konzepte und die Identifikation der gemeinsamen Subkonzepte, die unerfüllbar sind, sowie die Identifikation der beteiligten Korrespondenzen. Für das nähere Verständnis des Verfahrens wurde ein Beispiel eingeführt, das den Ablauf des Verfahrens verdeutlicht.

Um die Effektivität und die Effizienz des implementierten Verfahrens zu bestimmen, wurde eine ausführliche Evaluierung durchgeführt. Es wurde die beste Konfiguration für das implementierte Verfahren ermittelt. Hierfür wurden die GOMMA Mappings des Large BioMed Track verwendet. Mithilfe der identifizierten Konfiguration wurde das Verfahren mit den bereits existierenden Mapping Reparaturverfahren LogMap und Alcomo verglichen. Für den Vergleich der verschiedenen Verfahren wurden hinzufügend die Mappings von GOMMA-reuse, YAM++ und ServOMap verwendet. Alle Verfahren reduzieren die Anzahl der logischen Konflikte signifikant und erhöhen z.T. das F- Measure geringfügig. Das realisierte graphbasierte Verfahren übertrifft die Verfahren bzgl. der Effizienz und der logischen Konsistenz.

Für die zukünftige Arbeit soll die Methode der Reparatur des Mappings bzgl. der Beziehungen verbessert werden. Die bisherige Annahme der Schnittmengen der erweiterten Domain- und Range- Mengen ist nicht ausreichend, um eine gute Reparaturmenge zu identifizieren. Des

Weiteren sollen weitere Gewichtungsfunktionen untersucht werden, um die Qualität des resultierenden Mappings zu erhöhen. Diesbezüglich ist jedoch nicht anzunehmen, dass eine optimale Gewichtung existiert. So ist z.B. eine hohe Gewichtung der Ähnlichkeit nicht sinnvoll, wenn das zu reparierende Mapping von einem schlechten Matchverfahren generiert wurde. Um eine höhere Effizienz zu realisieren, soll das Verfahren parallelisiert werden.

Ein weiterer Aspekt neben der Steigerung der Effizienz und der Effektivität des Reparaturverfahrens ist das Zusammenspiel der einzelnen Teilprozesse des gesamten Match- Workflows. Konkret ist zu untersuchen, welchen Einfluss bestimmte Blocking- Verfahren oder bestimmte Matcher auf den Reparaturprozess haben und somit auf das resultierende Mapping. Ein weiterer Aspekt, der zu evaluieren ist, ist der Einsatz des Reparaturverfahrens während des Matchprozesses. Das Reparaturverfahren fungiert dabei als Reparatur und als Filterung, somit wird z.B. die Größe des Similarity- Cubes reduziert, der bei der GOMMA Match Komponente verwendet wird.

Ontologie- Mappings sind ein essentieller Bestandteil für Ontologie- Merging Verfahren. Ein inkorrektes Mapping als Eingabe für ein Ontologie- Merging Verfahren ist eine potentielle Ursache für eine fehlerhafte fusionierte Ontologie, die als Ergebnis des Merging Prozesses erzeugt wird. Jedoch existieren in der Praxis ebenfalls Ontologien, die logisch fehlerhaft modelliert sind, aufgrund einer fehlerhaften is_a Hierarchie. In der Zukunft sollen Ontologien bzgl. der Kohärenz untersucht werden und festgestellt werden, ob die Ursache der Inkohärenz bei fusionierten Ontologien stets ein inkorrektes Alignment ist. Diesbezüglich kann das graphbasierte Verfahren weiter entwickelt werden, so dass ebenfalls die Subsumptionen der Ontologie in den Reparaturprozess mit einbezogen werden, so dass als Ergebnis eine reparierte Ontologie bzgl. der is_a Hierarchie entsteht.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] O. Lassila and Deborah L. McGuinness. The role of frame-based representation on the semantic web. Technical Report KSL-01-02, Stanford University, Stanford, 2001.

[2] Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5):907–928, 1995.

[3] Salvatore Raunich and Erhard Rahm. Target-driven merging of taxonomies with atom. *Information Systems*, 42:1–14, 2014.

[4] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB JOURNAL*, 10:2001, 2001.

[5] Franz Baader and Werner Nutt. Basic description logics. In *Description logic handbook*, pages 43–95, 2003.

[6] Erhard Rahm. Towards large-scale schema and ontology matching. In *Schema matching and mapping*, pages 3–27. Springer, 2011.

[7] Fayçal Hamdi, Brigitte Safar, Chantal Reynaud, and Haïfa Zargayouna. Alignment-based partitioning of large-scale ontologies. In *Advances in knowledge discovery and management*, pages 251–269. Springer, 2010.

[8] Toralf Kirsten, Anika Gross, Michael Hartung, Erhard Rahm, et al. Gomma: a component-based infrastructure for managing and analyzing life science ontologies and their evolution. *J. Biomedical Semantics*, 2:6, 2011.

[9] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *IN SIGMOD CONFERENCE*, pages 906–908. ACM Press, 2005.

[10] Duy Hoa Ngo and Zohra Bellahsene. YAM++ : A multi-strategy based approach for Ontology matching task. In Andriy Nikolov Mathieu d'Aquin, editor, *Knowledge Engineering and Knowledge Management*, page 5, Galway City, Ireland, 2012.

[11] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *18th International Conference on Data Engineering (ICDE 2002)*, 2002.

[12] Daniel Faria, Catia Pesquita, Emanuel Santos, Matteo Palmonari, Isabel F. Cruz, and Francisco M. Couto. The agreementmakerlight ontology matching system. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Johann Eder, Zohra Bellahsene, Norbert Ritter, Pieter De Leenheer, and Deijing Dou, editors, *OTM Conferences*, volume 8185 of *Lecture Notes in Computer Science*, pages 527–541. Springer, 2013.

[13] Alexander K Hudek and Grant Weddell. Binary absorption in tableaux-based reasoning for description logics. In *Proc. of the 2006 Int. Workshop on Description Logics (DL 2006)*, volume 189, 2006.

[14] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In Alan Ruttenberg, Ulrile Sattler, and Cathy Dolbear, editors, *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, Karlsruhe, Germany, October 26–27 2008.

[15] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. Concurrent classification of $\mathcal{EL}$ ontologies. In *The Semantic Web–ISWC 2011*, pages 305–320. Springer, 2011.

[16] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. ELK reasoner: Architecture and evaluation. In *Proceedings of the OWL Reasoner Evaluation Workshop 2012 (ORE'12)*, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[17] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In *The Semantic Web–ISWC 2011*, pages 273–288. Springer, 2011.

[18] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267 – 284, 1984.

[19] Christian Meilicke. *Alignment incoherence in ontology matching*. PhD thesis, PhD thesis, University Mannheim, 2011.

[20] Ernesto Jiménez-Ruiz, Christian Meilicke, Bernardo Cuenca Grau, and Ian Horrocks. Evaluating mapping repair systems with large biomedical ontologies. In *Description Logics'13*, pages 246–257, 2013.

[21] E. Santos, D. Faria, C. Pesquita, and F. Couto. Ontology alignment repair through modularization and confidence-based heuristics. *ArXiv e-prints*, July 2013.

[22] Patrick Arnold. Semantic enrichment of ontology mappings: Detecting relation types and complex correspondences. In Kai-Uwe Sattler, Stephan Baumann, Felix Beier, Heiko Betz, Francis Gropengießer, and Stefan Hagedorn, editors, *Grundlagen von Datenbanken*, volume 1020 of *CEUR Workshop Proceedings*, pages 34–39. CEUR-WS.org, 2013.

[23] O. Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 32(suppl 1):D267–D270, 2004.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Ort, Datum	Unterschrift