Inkrementelle Materialisierung von Ontologien basierend auf Beschreibungslogik

Incremental Materialization of Description Logic Ontologies

Björn Kreutz

Universität Koblenz-Landau bjoernkreutz@uni-koblenz.de

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Zusammenfassung Das vorausberechnen von implizitem Wissen nennt sich Materialisierung. Dieser Vorgang vereinfacht den schnellen Zugriff auf Daten, ist jedoch auch aufwendig zu erstellen. Diese Bachelorarbeit stellt eine Software vor, die auf einer Menge von beschreibungslogischen Axiomen eine Materialisierung erzeugt und diese, bei Änderungen in der Ontologie, inkrementell wartet, indem sie Schlussfolgerungsketten aufzeichnet und logische Abhängigkeiten verwaltet. Dieses Vorgehen sorgt dafür, dass die Materialisierung nach einer initialen Berechnung ohne großen Rechenaufwand aktualisiert werden kann. Eine im Rahmen dieser Arbeit durchgeführte Laufzeitanalyse zeigt, dass dieser *inkrementelle* Ansatz weitaus schneller Daten in die Materialisierung integriert als ein naiver Ansatz.

Abstract The precomputation of implicit knowledge is called materialization. This process enables the quick access to derived knowledge, but is expensive to create. This paper presents a software that derives a materialization from a set of description logic axioms, and maintains it incrementally throughout it's runtime by tracking the performed derivations and logical dependencies. This approach enables the maintenance of the materialization, without the need to rederive the entire set. The runtime analysis performed in this paper shows the resulting increase in performance, and highlights the advantage of this incremental approach compared to a naive algorithm.

1 Einleitung

Materialisierungen werden schon länger in logischen Datenbanken verwendet um die benötigten Rechenoperationen pro Anfrage zu minimieren. Nach einer vergleichsweise aufwendigen anfänglichen Berechnung sind die Antworten auf die meisten Anfragen in der Materialisierung enthalten und können sofort ausgelesen werden. Dies ist vor allem bei umfangreichen und statischen Datenbanken sinnvoll, da diese die Materialisierung längere Zeit verwenden können und der Rechenaufwand pro Anfrage geringer ist. Materialisierungen bieten sich zum einen in Datenbanken an, die große Mengen an Daten verwalten müssen, denn mit einer materialisierten Ansicht lassen sich relevante Daten leichter Auslesen. Zum anderen werden Materialisierungen benutzt, wenn eine schnelle Laufzeit von Nöten ist. Dies trifft zum Beispiel bei der KI Programmierung zu. In diesem Feld ist es nötig, überflüssige logische Operationen zu vermeiden, welche die Laufzeit verlangsamen.

Das Hauptproblem, welches bei der Materialisierung von logischen Datenbanken auftritt, ist die materialisierte Ansicht aktuell zu halten. Ontologien sind in den wenigsten Fällen über längere Zeit komplett statisch, und schon kleinere Änderungen beeinflussen die Konsistenz der Materialisierung. Wenn keine speziellen Vorkehrungen getroffen werden, muss bei jeder Veränderung in der Ontologie die gesamte Materialisierung erneut berechnet werden, was Zeit und Ressourcen kostet. Diese Vorgehensweise wird in der vorliegenden Arbeit als der *naive* Ansatz bezeichnet.

Der in dieser Arbeit umgesetzte Ansatz zur Materialisierung erlaubt es, die von Änderungen in der Ontologie betroffenen Elemente zu isolieren, anzupassen und in die bereits existierende Materialisierung zu integrieren. Auf diese Weise kann eine errechnete Materialisierung angepasst werden und muss nicht vollständig neu berechnet werden. In der vorliegenden Arbeit wird dieser Ansatz als *inkrementell* bezeichnet.

Die Arbeit ist in 5 Kapitel unterteilt. Die Einleitung bildet Kapitel 1. In Kapitel 2 werden die benutzten Grundlagen erläutert. Kapitel 3 beschreibt die Architektur und Implementation der Software, sowie die erarbeiteten Algorithmen. In Kapitel 4 wird eine Laufzeitanalyse durchgeführt, die den inkrementellen Ansatz mit einem naiven Ansatz vergleicht. Kapitel 5 fasst die gewonnenen Erkenntnisse zusammen.

2 Grundlagen und Hintergrund

In diesem Kapitel werden die grundlegenden Konzepte erläutert, die in dieser Arbeit verwendet werden.

Der erste Abschnitt beschreibt einige Grundlagen zu Beschreibungslogiken und erläutert den allgemeinen Aufbau anhand einiger kurzer Beispiele. Der zweite Abschnitt erläutert kurz das Prinzip der Materialisierung und welche konkreten Aspekte bei einer Materialisierung beachtet werden müssen. Der dritte Abschnitt zeigt einige ähnliche Arbeiten auf und setzt diese Arbeit in Kontext.

2.1 Beschreibungslogik

Beschreibungslogiken sind formale Sprachen, die zur Wissensrepräsentation genutzt werden. Sie finden in vielen Teilgebieten der Informatik Anwendung, besonders allerdings auf dem Gebiet des Semantic Web. Im Bereich der Beschreibungslogiken bedeutet *Wissensrepräsentation*, dass Wissen über Einheiten und ihre Beziehung, wie zum Beispiel verschiedene Tierarten, formal dargestellt und maschinenlesbar gemacht wird, was das Verarbeiten von komplexen logischen Zusammenhängen ermöglicht. So können nicht nur logische Zusammenhänge effizient formuliert werden, sondern auch logische Schlussfolgerungen getroffen werden.

2.1.1 Grundlegende Elemente

Beschreibungslogiken bestehen im allgemeinen aus Konzepten, Rollen und Individuen.

Konzepte repräsentieren Klassen von Individuen. Sie sind Sammelbegriffe für Individuen. In First-Order-Logic wird der Begriff *Klasse* benutzt, was sich auch in OWL[1] durchgesetzt hat. In dem bereits erwähnten Beispiel der Tierarten wäre unter anderem *Katze* ein Konzept welches eine Sammlung von individuellen Tieren repräsentiert.

Rollen repräsentieren binäre Relationen zwischen Individuen. Falls nicht anders definiert, werden sie allgemein als einseitig gerichtete Beziehungen aufgefasst. In OWL und First-Order-Logic wird dies mit *Prädikat* oder *Eigenschaft* bezeichnet, ist allerdings effektiv äquivalent. Ein Beispiel im Kontext der Tierarten wäre *mag*, *nachbarVon*, oder *vaterVon*.

Individuelle Namen bezeichnen einzelne Individuen. Sie können in Rollenrelationen zueinander stehen und Klassen instanziieren. Der Name *Mittens* zum Beispiel beschreibt ein Individuum. Beschreibungslogiken machen im Allgemeinen keine UNA[2] was bedeutet, dass verschiedene Namen dasselbe Individuum bezeichnen können, sofern dies nicht explizit verneint wird. Dies muss besonders beachtet werden, wenn zum Beispiel Wissen über dasselbe Gebiet aus verschiedenen Quellen zusammengelegt wird. [KSH12]

[1] Web Ontology Language
[2] Unique Name Assumption

2.1.2 Knowledge Base

Definition 1. *Eine Knowledge Base (KB)* ***K*** *besteht aus der Menge aller TBox-Axiome* ***T*** *und aller ABox-Axiome* ***A*** *und umfasst die gesamte Menge aller Axiome in einer Ontologie. Somit gilt* ***K=(T,A)****.*

Axiome sind die grundlegenden Modellierungskonzepte in Beschreibungslogiken. Sie sind Regeln die beschreiben, in welcher Relation spezifische Konzepte, Rollen und Individuen stehen. Sie müssen stets für den beschriebenen Zustand der Ontologie gelten. Abhängig davon, welche logischen Zusammenhänge ein Axiom beschreibt wird es verschiedenen Kategorien zugeordnet. Diese jeweiligen Regelmengen werden als „Box“ bezeichnet. Jede Beschreibungslogik besitzt eine *ABox* und eine *TBox*.

Die **TBox** (*Terminological Box*) enthält alle Axiome, die Relationen zwischen Konzepten beinhalten. Ein solches Axiom ist das Subsumptionsaxiom. Es beschreibt, dass ein Konzept ein Unterkonzept eines anderen Konzeptes ist. Im Beispiel der Tierarten gilt, dass alle *Katzen Säugetiere* sind, somit gilt das folgende Subsumptionsaxiom.

$$Katze \sqsubseteq Säugetier \tag{1}$$

Katze ist somit ein Unterkonzept von Säugetier und jedes Individuum das Katze repräsentiert, repräsentiert auch Säugetier. Allgemein lässt sich festlegen, dass die TBox alle Axiome enthält, die keine Individuen beschreiben. Die TBox beschreibt schematisches und strukturelles Wissen.

Die **ABox** (*Assertional Box*) hingegen enthält jene Axiome, die Relationen von Individuen beschreiben und zu welchen Konzepten sie gehören. Die einfachste Form des ABox-Axioms ist eine *Concept Assertion*, das heißt ein Axiom welches sicherstellt, dass ein Individuum einem Konzept angehört. In dieser Arbeit wird der deutsche Begriff *Konzeptzuweisung* verwendet werden. Der Term 2 stellt sicher, dass ein Individuum *Mittens* das Konzept *Katze* repräsentiert. *Mittens* ist demnach eine Instanz des Konzeptes *Katze*.

$$Mittens : Katze \tag{2}$$

Des weiteren enthält die ABox auch *Role Assertions*, welche in dieser Arbeit mit dem Begriff *Rollenzuweisungen* übersetzt werden. Eines solches Axiom beschreibt Relationen von zwei benannten Individuen. Das Axiom

$$(Mittens, Tazzie) : mag \tag{3}$$

beschreibt beispielsweise, dass *Mittens* ein Individuum namens *Tazzie* mag. Die ABox enthält somit Faktenwissen . [KSH12] [BN03]

2.1.3 Boolsche Konstruktoren

Mit diesen grundlegenden Axiomen können schon Rollenverhältnisse modelliert werden. Um komplexere Zusammenhänge darzustellen, werden allerdings zusätzliche Operatoren benötigt. Beschreibungslogiken erlauben das Definieren von Konzepten über Operatoren, die funktional den Operatoren aus der Booleschen Algebra entsprechen.

Die *Konjunktion* $\sqcap$ repräsentiert das logische *und*. Dies wird auch als die *Schnittmenge* bezeichnet. In der Mengenlehre sind nur die Elemente, die sich in beiden Mengen befinden, auch in der resultierenden Menge enthalten. Analog dazu sind in einer Konjunktion von Konzepten genau die Elemente enthalten, die in beiden Konzepten enthalten waren.

$$Schnabeltier \equiv Säugetier \sqcap Eierleger \tag{4}$$

In diesem Beispiel wird das Konzept Schnabeltier darüber definiert, dass es genau all die Elemente enthält, die sowohl zu Säugetieren als auch zu Eierlegern zählen.

Die *Disjunktion* $\sqcup$ entspricht dem logischen *oder* und wird auch als *Vereinigung* bezeichnet. Analog zur Mengenlehre enthält ein Konzept, welches über die Disjunktion zweier Konzepte C und D definiert ist, alle Elemente die in C oder in D auftauchen, wobei hier auch, wie in der Mengenlehre, kein exklusives *oder* gemeint ist.

$$Katze \sqcup Hund \tag{5}$$

Dieses Beispiel beschreibt die Summe der Tiere die Hunde oder Katzen sind.

Die *Negation* $\neg$, oder auch das *Komplement*, wird benutzt um zu signalisieren, dass eine Menge von Individuen *nicht* zu einem Konzept gehört. Das Konzept $\neg Säugetier$ beispielsweise beschreibt alle Individuen, die nicht Säugetiere sind.

In Kombination mit dem Konjunktionsoperator $\sqcap$ und dem Disjunktionsoperator $\sqcup$ können erweiterte Konzepte definiert werden.

$$Pinguin \equiv Vogel \sqcap \neg Flieger \tag{6}$$

(6) demonstriert die Verwendung des Konjunktionsoperators in Kombination mit der Negation. Es definiert alle Individuen, die Vögel sind und nicht fliegen, als Pinguine.

Die Konzepte *Top* $\top$ und *Bottom* $\bot$ repräsentieren allgemeine Konzepte. *Top* stellt das allgemeingültige Konzept dar, das heißt es enthält jedes mögliche Individuum als Instanz.

$$\top \sqsubseteq male \sqcup female \tag{7}$$

(7) beschreibt die Disjunktion der beiden Geschlechter männlich und weiblich und setzt Top als ihr Unterkonzept. Dies bedeutet, dass jedes Individuum im betrachteten Universum aus einer Teilmenge der männlichen oder der weiblichen Individuen stammt und somit kein Individuum ohne Geschlecht existiert.

Das Konzept *Bottom* $\bot$ ist das Gegenstück zu *Top*. $\bot$ beschreibt das leere Konzept, das heißt das Konzept, das keine Instanzen besitzt.

$$male \sqcap female \sqsubseteq \bot \tag{8}$$

Das Axiom (8) modelliert die Konjunktion der beiden Geschlechter als Unterkonzept von $\bot$. Diese Aussage lässt sich so interpretieren, dass es kein Individuum gibt, das sowohl männlich als auch weiblich ist.

Rollenrestriktionen sind ein weiterer Weg Konzepte zu definieren. Hier kommen die All- und Existenzquantoren ins Spiel. Allgemein gefasst definieren diese Operatoren, ob Relationen zu bestimmten Konzepten bestehen.

In Beschreibungslogiken wird der *Allquantor* in der Form $\forall R.C$ verwendet, wobei R eine beliebige Rolle und C ein beliebiges Konzept ist. Allgemein bezeichnet dies alle Elemente die zu C in der Relation R stehen.

$$\forall hatHaustier.Katze \tag{9}$$

Der Ausdruck (9) kann gelesen werden als *Individuen, die in allen Relationen hatHaustier eine Katze haben*. Umgangssprachlich bezeichnet der Ausdruck die Menge alle Individuen, die als Haustiere nur Katzen besitzen. Wichtig hier ist allerdings zu bemerken, dass das Axiom auch Individuen beinhaltet, die *keine* Haustiere haben. Der Reasoner filtert lediglich alle Individuen heraus, die andere Haustiere außer Katzen haben. Eine weitere Möglichkeit (9) darzustellen wäre $\neg\exists hatHaustier.\neg Katze$, das heißt es existieren keine Individuen, die ein Haustier haben und dieses Haustier nicht als Katze zählt.

Seien R eine Rolle und C ein Konzept, so beschreibt $\exists R.C$ alle Individuen, die mindestens ein C in der Relation R haben.

$$\exists hatHaustier.Katze \tag{10}$$

Im Gegensatz zu (9) beschreibt (10) alle Individuen, die in der Menge ihrer Haustiere mindestens eine Katze haben. Abhängig davon, welche Zusammenhänge modelliert werden muss beachtet werden, dass das Modell endlich bleibt. Sei ein Mensch definiert durch $Mensch \equiv \exists hatVorfahr.Mensch$, das heißt jeder Mensch stammt von einem Menschen ab. Unter Verwendung eines naiven Lösungsalgorithmus läuft die Berechnung endlos rekursiv, da von jedem Menschen auf einen weiteren unbekannten Menschen geschlossen werden kann. Moderne Tableau-Verfahren sind soweit optimiert, dass sie diese Schlussfolgerungen an einem bestimmten Punkt blocken.

Mithilfe dieser Operatoren können nun auch wiederum Konzepte definiert werden.

$Katzenbesitzer \equiv \exists hatHaustier.\top \sqcap \forall hatHaustier.Katze$ bezeichnet das Konzept eines Katzenbesitzers, ein Individuum das mindestens ein Haustier hat und alle Haustiere Katzen sind. [KSH12] [BN03]

2.1.4 Semantik

Im Folgenden Abschnitt wird die Semantik der Beschreibungslogiken konkretisiert.

Konzepte werden semantisch allgemein als Mengen von Individuen definiert. Jedes Individuum das in solch einer Menge enthalten ist kann als Instanz dieses Konzeptes aufgefasst werden.

Definition 2. *Sei ein Konzept* $\boldsymbol{C}$ *und eine Instanz in diesem Konzept* $\boldsymbol{a}$*, so gilt* $C(a) \equiv a \in C$ *Ein Individuum a ist vom Konzept C genau dann wenn a Element der Menge C ist.*

Rollen sind modelliert als Mengen, die geordnete Paare von Individuen enthalten. Jedes Tupel repräsentiert eine gerichtete Beziehung zwischen diesen beiden Individuen.

Definition 3. *Sei r eine Relation und a und b Individuen. a und b stehen in Relation r zueinander, genau dann wenn das Tupel a,b Element der Menge r ist. Somit gilt* $(a, b) : r \equiv (a, b) \in r$.

Alle *Individuen* stammen aus einer gegebenen Domäne.

Die *Signatur* einer Beschreibungslogik wird beschrieben durch (N_C, N_R, N_O), das heißt die Menge aller Klassen, Relationen und Objekte. Eine *Terminologische Interpretation* $I = (\Delta^I, *^I)$ auf dieser Signatur besteht somit aus einer nicht-leeren Domäne Δ^I und einer Interpretationsfunktion $*^I$. Diese Interpretationsfunktion bildet die folgenden Elemente der Beschreibungslogik auf bestimmte Interpretationen ab. Jedes Individuum a wird auf ein Element $a^I \in \Delta^I$ abgebildet. Jedes *Konzept* wird auf eine Untermenge der Domäne Δ^I abgebildet. Jede Rolle auf eine Untermenge von $\Delta^I \times \Delta^I$. [BN03]

Definition 4. *Die Semantik der einzelnen Befehle ist wie folgt definiert.*

$\top^I = \Delta^I$ *Die Interpretation von* $\top$ *umfasst die gesamte Domäne.*

$\bot = \emptyset$ *Die Interpretation von* $\bot$ *ist immer leer.*

$(C \sqcup D)^I = C^I \cup D^I$.

$(C \sqcap D)^I = C^I \cap D^I$.

$(\neg C)^I = \Delta^I \backslash C$ *Die Interpretation der Negation eines Konzeptes entspricht allen Elemente, ausgenommen jene die im Konzept auftreten.*

$(\forall R.C)^I = \{x \in \Delta^I | \text{für jedes } y, (x,y) \in R^I \text{ impliziert } y \in C^I\}$.

$(\exists R.C)^I = \{x \in \Delta^I | \text{es existiert } y, (x,y) \in R^I \text{ und } y \in C^I\}$.

Definition 5. *Die TBox und ABox sind auf folgende Weise modelliert.*

TBox:

$I \vDash C \sqsubseteq D$, *nur genau dann wenn* $C^I \subseteq D^I$.

$I \vDash T$ *wenn* $I \vDash \Phi$ *für jedes* $\Phi \in T$.

ABox:

Konzeptzuweisung $I \vDash a : C$, *falls* $a^I \subseteq C^I$.

Relation $I \vDash (a,b) : R$, *falls* $(a^I, b^I) \in R^I$.

$I \vDash A$, *falls* $I \vDash \phi$ *für jedes* $\phi \in A$.

2.1.5 Tableau-Algorithmus

Der Tableau-Algorithmus wird auf eine Knowledge Base $K = (T, A)$ angewendet und überprüft ihre Korrektheit, das heißt ob sie Widersprüche enthält, oder erfüllbar ist. Im Falle der Erfüllbarkeit wird ein widerspruchsfreies Modell erzeugt.

Die KB muss vorher in die *Negationsnormalform NNF* transformiert werden. Dies bedeutet, dass Negationssymbole nur vor atomaren Elementen stehen dürfen. Jedes Axiom lässt sich mit Hilfe der De Morganschen Regeln in solch eine Form bringen. Der Algorithmus arbeitet auf einem Datenformat namens *Completion Forest*. Basierend auf spezifischen Erweiterungsregeln wird der Algorithmus genutzt, um die KB in eine Baumstruktur zu transformieren.

Für jede Operation gibt es eine sogenannte *Expansion Rule*, das heißt Regeln die bei bestimmten logischen Ausdrücken zum Einsatz kommen und diese ersetzen, beziehungsweise auflösen. Die *Expansion Rules* nach Baader [BS01] sind wie folgt formuliert.

Definition 6. *Die **Expansion Rules** nach Baader [BS01]:*

Die* $\rightarrow \sqcap$*-Regel
Bedingung: A enthält $(C_1 \sqcap C_2)(x)$*, und nicht sowohl* $C_1(x)$ *als auch* $C_2(x)$*.*
Aktion: A':= $A \cup \{C_1(x), C_2(x)\}$ *.*

Die* $\rightarrow \sqcup$*-Regel
Bedingung: A enthält $(C_1 \sqcup C_2)(x)$*, und weder* $C_1(x)$ *noch* $C_2(x)$*.*
Aktion: A':= $A \cup \{C_1(x)\}$ *,A":=* $A \cup \{C_2(x)\}$ *.*

Die* $\rightarrow \exists$*-Regel
Bedingung: A enthält $(\exists r.C)(x)$*, und es existiert kein Individuum z sodass* $C(z)$ *und* $r(x, z) \notin A$ *.*
Aktion: A':= $A \cup \{C(y), r(x, y)\}$*, wobei* $y \notin A$*.*

Die* $\rightarrow \forall$*-Regel
Bedingung: A enthält $(\forall r.C)(x)$ *und r(x,y), aber nicht C(y).*
Aktion: A':= $A \cup \{C(y)\}$*.*

Der nächste Erweiterungsschritt, und somit die anzuwendende Regel, wird mit einem Don't-Care-Nichtdeterminismus ausgewählt, was bedeutet, dass die Reihenfolge der Auflösungsschritte zwar die Rechenzeit beeinflussen kann, auf das Ergebnis jedoch keinen Einfluss hat und kein backtracking benötigt.

Besonders rechenaufwendig ist die $\sqcup$-Regel. Sie basiert auf einem Don't-Know-Nichtdeterminismus, dies bedeutet die nichtdeterministische Wahl zwischen zwei Termumformungsmöglichkeiten. Da nur eine Möglichkeit terminieren muss um Konsistenz aufzuzeigen, es allerdings nicht abzusehen ist welche der Umformungen dazu führt, benötigt der Algorithmus gegebenenfalls backtracking um im Falle des Scheiterns des ersten Pfades den zweiten Pfad zu analysieren.

Die Nutzung von TBoxen kann allerdings zu endlosen Auflösungsbäumen führen. Das Axiom $Mensch \sqsubseteq \exists hatMutter.Mensch$ definiert jeden Menschen dadurch, dass er/sie eine menschliche Mutter hat. Diese undefinierte Mutter ist allerdings auch menschlich und muss folglich auch eine menschliche Mutter haben. Dies erzeugt eine endlose Schlussfolgerungskette. Um dies zu vermeiden, müssen bestimmte Äste des Auflösungsbaumes geblockt werden. Dieses *Blocking* wird ausgelöst, sobald ein Knoten, der durch einen Existenzquantoren neu erzeugt wurde, eine Untermenge seines Oberknoten bildet. Sobald ein Knoten geblockt wurde, wird er nicht weiter aufgelöst. [BS01]

In der vorliegenden Arbeit wird ein Schlussfolgerungsalgorithmus zur Materialisierung verwendet, dessen Rechenschritte auf den *Expansion Rules* des Tableau-Kalküls basieren. Die expliziten Umformungen sind näher in 3.3.2 erläutert.

2.1.6 Konkrete Beschreibungslogiken

Dieser Abschnitt stellt kurz zwei wichtige Beschreibungslogiken dar, die in der vorliegenden Arbeit behandelt werden.

AL steht für Attributive Language. AL ist eine grundlegende Form der Beschreibungslogik auf der viele ausdrucksstärkere Logiken aufbauen. Oft wird allerdings nicht die reine AL, sondern ihre um Konzeptnegation erweiterte Form ALC verwendet. Seien C und D Konzepte und R eine Rolle, es gilt die folgende Form. [Zol13]

$$ALC ::= \bot|\top|A|\neg C|C \sqcap D|C \sqcup D|\exists R.C|\forall R.C \tag{11}$$

(11) zeigt die grundlegenden Operationen die in ALC möglich sind. Trivialerweise enthält sie die Konzepte $\top$, $\bot$ und jedes atomare Konzept A aus der Menge aller Konzepte. ALC ermöglicht außerdem das Bilden von Konzeptschnittmengen ($\sqcap$) und Konzeptvereinigungen ($\sqcup$), Existenzquantifikation ($\exists$) und Allquantifikation ($\forall$) auf Rollen in einem Konzept, und Komplemente von allen Konzepten ($\neg$).

ALC wird als eine der grundlegenden Beschreibungslogiken angesehen und kann um bestimmte Befehle erweitert werden um komplexere Logiken zu bilden. Es gibt einige Erweiterungen, die auf ALC angewendet werden können. Zu diesen zählt unter anderem die Kardinalitätsrestriktion, welche allgemein mit einem **N** dargestellt wird. Diese Erweiterung erlaubt es, die genaue Anzahl der Rollenverhältnisse zu definieren und abzufragen. Es gibt einige Erweiterungen, welche die Verwaltung von Rollen betreffen. Diese Relationen beinhalten Rollentransitivität , Rollenhierarchie oder sogar komplexe Rolleninklusion. Diese drei Eigenschaften werden jeweils für gewöhnlich durch ein **S**, **H** und **R** im Namen der Beschreibungslogik dargestellt. So wird eine ALC mit Rollenhierarchie als ALCH bezeichnet, was intuitiv die modulare Namenskonvention aufzeigt. Es gibt allerdings Ausnahmen bei der Bezeichnung, so wird eine ALC mit Rollentransitivität allgemein lediglich als ***S*** bezeichnet, was wiederum als Grundlage für Beschreibungslogiken wie beispielsweise **SHOIN(D)** dient, welche in OWL-DL verwendet wird. [Zol13] [Lie10]

EL ist eine leichtgewichtigere Form der Beschreibungslogik. Sie wurde entwickelt um Ontologien zu verwalten, die große Mengen an wenig komplexen Daten beinhalten. Dies findet vor allem Anwendung in den Biowissenschaften, wo große Mengen terminologischer Daten verwaltet werden müssen.

$$EL ::= \top|A|C \sqcap D|\exists R.C \tag{12}$$

Wie in (12) zu sehen ist erlaubt EL unbegrenzte Nutzung des Existenzquantors ($\exists$) und Konzept Schnittmengen ($\sqcap$) . Im Gegensatz zu ALC ermöglicht EL keine Schnittmengen ($\sqcup$) , Komplemente ($\neg$) oder Allquantoren ($\forall$).

EL, und Erweiterungen von EL, wie zum Beispiel EL++ können in Polynomialzeit gelöst werden. Wie bereits im Tableau-Algorithmus erwähnt, entsteht ein Großteil des Rechenaufwandes durch die Erweiterungsregel welche die Disjunktion betrifft. Diese $\sqcup$-Regel ist im Tableau-Algorithmus nicht deterministisch und benötigt somit mehr Rechenressourcen. Da EL keine Disjunktionen erlaubt, kann ein Tableau-Algorithmus effizienter ausgeführt werden. ALC ist zwar mächtiger als EL, erlaubt jedoch lediglich eine Laufzeit von exponentieller Zeit im worst-case. [KSH12]

Da sich beide Sprachen lediglich in der Anzahl der erlaubten Befehle unterscheiden, ist es möglich, die Verarbeitung modular aufzubauen. So können Algorithmen zuerst für EL entwickelt werden und dann um Befehle aus ALC erweitert werden. Der im Rahmen dieser Ausarbeitung erstellte Prototyp verwendet einen solchen modularen Aufbau und kann deshalb sowohl für EL als auch ALC verwendet werden, sowie gegebenenfalls mit weiteren Befehlen ausgebaut werden.

2.2 Materialisierung

In diesem Abschnitt werden die Grundbegriffe und Ansätze zu Materialisierungen erläutert.

Materialisierung bezeichnet im Allgemeinen die Idee, abgeleitetes Wissen auf einer Ontologie im Voraus zu berechnen. Der in der vorliegenden Arbeit verwendete Begriff *Materialisierung* leitet sich aus dem im englischen verwendeten Begriff *materialized view* ab und beschreibt einen Datensatz, der aus den gegebenen Grunddaten zu einem bestimmten Zeitpunkt abgeleitet wurde um bestimmte Sachverhalte deutlicher darzustellen. Dies kann zum Beispiel implizit in den Daten enthaltenes Wissen sein und ermöglicht das Arbeiten auf diesen Daten ohne ständiges Abfragen der zugrundeliegenden Daten. Da Datenbanken für gewöhnlich nicht komplett statisch sind, sondern gelegentlich aktualisiert werden müssen, macht es Sinn die Aktualisierungsvorgänge zu optimieren. [GM99]

Im Kontext dieser Arbeit wird zwischen verschiedenen Arten von Wissen unterschieden. Allgemein wird in einer Datenbank zwischen extensionalem und intensionalem Wissen unterschieden. Extensionales Wissen umfasst das Wissen, welches in der *ABox* explizit definiert wird. Diese Form von Wissen listet Elemente und ihre Relationen auf und wird für gewöhnlich als eine Menge von Einträgen geschrieben. Diese Menge wird als *Extension eines Prädikaten* bezeichnet. Angenommen *mag* ist ein Prädikat, dann gilt Beispielsweise

$$mag = \{(Mittens, Tazzi), (Mittens, Franzi)\} \tag{13}$$

Die Extension von mag ist in 13 angegeben. Sie wird repräsentiert durch eine Liste von Einträgen die alle Freundschaften in der Ontologie aufzählt.

Intensionales Wissen wird in der *TBox* definiert. Es beinhaltet Regeln, die Aussagen über die logischen Zusammenhänge treffen. Hierunter fallen solche Axiome, die zum Beispiel Transitivität oder Verhältnisse von Konzepten definieren. Intensionales Wissen betrifft allgemein Schemata und Regeln, und enthält keine Individuen.

Generell können Materialisierungen jede Form des impliziten Wissens beinhalten. Dieses Wissen wird, um Rechenzeit zu sparen, explizit formuliert und separat abgespeichert. Hierbei kann allerdings, abhängig davon welche Art von Wissen materialisiert wird, sowohl die anfängliche Materialisierung, als auch der Performancegewinn beim Abfragen der Materialisierung stark variieren.

Allgemein lässt sich eine Materialisierung erstellen, indem zuerst die gesamte Extension jeder Rolle und jeder Klasse errechnet wird. Dies bedeutet, dass für jede Klasse C eine Liste C^I erstellt wird, die alle Instanzennamen enthält die C repräsentieren, und für jede Rolle R eine Liste R^I erstellt wird die alle Individuenpaare enthält, die in dieser Relation stehen. Diese Listen können schon teilweise explizit in der ABox definiert sein, der Sinn hinter einer Materialisierung ist jedoch, dass der Großteil noch aus der Ontologie abgeleitet werden muss.

Von einer gegebenen Menge ABox- und TBox-Axiome lassen sich, analog zum bereits erläuterten Tableau-Algorithmus, weitere Axiome herleiten. Im Folgenden wird vorerst grob erläutert werden, wie dies vonstattengeht.

Der erste Fall tritt auf, wenn von einem TBox-Axiom auf ein weiteres TBox-Axiom geschlossen werden kann. Seien C,D und E Konzepte und $C \sqsubseteq D \sqcap E$ ein Axiom. So

lässt sich analog zum Tableau-Algorithmus herleiten dass $C \sqsubseteq D \sqcap E \rightarrow C \sqsubseteq D, C \sqsubseteq E$ gilt. Auf dieselbe Weise lässt sich auch aus ein Individuen-Axiom aus der ABox erweitern. Sei a ein Indiviuum, C und D Konzepte und $a : C \sqcap D$ ein Axiom in der ABox, so gilt $a : C \sqcap D \rightarrow a : C, a : D$. Des Weiteren lassen sich neue ABox-Axiome durch die Kombination von ABox- und TBox-Axiomen errechnen. Sei $a : C$ und $C \sqsubseteq D$ je ein Axiom, so gilt $a : C * C \sqsubseteq D \rightarrow a : D$. Das neu errechnete Axiom $a : D$ bezeichnet das Individuum a nun explizit als Vertreter von D, eine Information die zwar implizit aus den beiden anfänglichen Axiomen hervorging, nun jedoch explizit formuliert wurde. Diese Umformungen sind für alle Operatoren in ALC anwendbar und zerlegen ein Axiom analog zu den Regeln des Tableau-Algorithmus.

An dieser Stelle ist hervorzuheben, dass C und D zwar Konzepte sind, die Verknüpfung von Konzepten jedoch auch Konzepte erzeugt. C und D können somit jede beliebige Verschachtelung von Konzepten darstellen. Somit sind alle Umformungen induktiv definiert und können solange angewandt werden bis kein neues Wissen abgeleitet werden kann.

Im Kapitel 3.3.2 werden die konkreten Umformungen für die einzelnen Konzeptarten genauer erläutert und in den Kontext dieser Arbeit eingebracht.

Eine weitere Möglichkeit implizites Wissen abzuleiten besteht in Rolleneigenschaften. Einer Rolle kann eine Eigenschaft übergeben werden, die auf weitere Individuenpaare in dieser Rolle schliessen lassen kann. Solche Eigenschaften sind unter anderem *Reflexivität*, *Symmetrie* oder *Transitivität*. Da diese Arbeit sich allerdings nur mit **ALC** beschäftigt, sei diese Art von Materialisierung zwar erwähnt, allerdings nicht weiter im Detail ausgeführt.

Die folgende Tabelle 1 beschreibt den Anwendungsfall einer Materialisierung die auf Beschreibungslogik basiert. Die Spalte Input enthält die Axiome die hinzugefügt oder entfernt werden, wobei das Hinzufügen und Entfernen jeweils mit + und - gekennzeichnet ist.

Die ersten drei Axiome die hinzugefügt werden sind simple ABox-Axiome. Sie haben keinen Einfluss auf die TBox und enthalten kein ableitbares Wissen. Das vierte Axiom beschreibt das Verhältnis zweier Konzepte und gehört somit in die TBox. Dieses Axiom gilt für alle Individuen, die dem Konzept angehören welches sich im Kopf des Axioms befindet, in diesem Fall *a*. Daraus folgt, dass *a:B*. Schritt 5 ist ein weiteres TBox-Axiom und führt zu weiterem abgeleitetem Wissen.

Zeile 6 entfernt nun ein ABox-Axiom aus der Materialisierung. Wichtig ist hier zu bemerken, dass die Existenz von a durch die Relation r(b,a) noch bekannt ist, die Konzeptzugehörigkeit allerdings nicht mehr hergeleitet werden kann und somit nicht separat in der Materialisierung auftaucht. Durch das Hinzufügen eines weiteren TBox-Axioms in Zeile 7 kann a:A jedoch implizit hergeleitet werden. In Zeile 8 wird ein TBox-Axiom entfernt, was Auswirkungen auf das abgeleitete Wissen hat und dazu führt, dass 3 Einträge in der Materialisierung gelöscht werden.

Zeile 9 führt ein TBox-Axiom mit einer Konjunktion ein, was dazu führt, dass das Individuum d um zwei Konzeptzugehörigkeiten erweitert wird. Zeile 10 demonstriert, wie durch das Löschen einer Relation ein Individuum aus der Materialisierung entfallen kann. Zeile 11 und 12 etablieren eine alternative Herleitung für *d:D*. Daraufhin wird in Zeile 13 die ursprüngliche Herleitung gelöscht, da allerdings eine alternative Herleitung existiert, bleibt *d:D* erhalten.

Tabelle 1. Beispiel Materialisierung

-1.) Input	ABox	TBox	Materialisierung
0.) -	-	-	-
1.) +a:A	a:A	-	a:A
2.) +b:B	a:A,b:B	-	a:A, b:B
3.) +(b,a):r	a:A, b:B, (b,a):r	-	a:A, b:B, (b,a):r
4.) $+A \sqsubseteq B$	a:A, b:B, (b,a):r	$A \sqsubseteq B$	a:A, a:B, b:B, (b,a):r
5.) $+B \sqsubseteq \exists s.C$	a:A, b:B, (b,a):r	$A \sqsubseteq B$, $B \sqsubseteq \exists s.C$	a:A, a:B, (a,e):s, b:B, (b,a):r, (b,d):s, d:C, e:C
6.) -a:A	b:B, (b,a):r	$A \sqsubseteq B$, $B \sqsubseteq \exists s.C$	b:B, (b,a):r, (b,d):s, d:C
7.) $+B \sqsubseteq \forall r.A$	b:B, (b,a):r	$A \sqsubseteq B$, $B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$	a:A, a:B, (a,e):s, b:B, (b,a):r, (b,d):s, d:C, e:C
8.) $-A \sqsubseteq B$	b:B, (b,a):r	$B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$	a:A, b:B, (b,a):r, (b,d):s, d:C
9.) $+C \sqsubseteq D \sqcap E$	b:B, (b,a):r	$B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$, $C \sqsubseteq D \sqcap E$	a:A, b:B, (b,a):r, (b,d):s, d:C, d:D, d:E
10.) -(b,a):r	b:B	$B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$, $C \sqsubseteq D \sqcap E$	b:B, (b,d):s, d:C, d:D, d:E
11.) +(b,d):t	b:B, (b,d):t	$B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$, $C \sqsubseteq D \sqcap E$	b:B, (b,d):s, (b,d):t, d:C, d:D, d:E
12.) $+B \sqsubseteq \forall t.D$	b:B, (b,d):t	$B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$, $C \sqsubseteq D \sqcap E$, $B \sqsubseteq \forall t.D$	b:B, (b,d):s, (b,d):t, d:C, d:D, d:E
13.) $-C \sqsubseteq D \sqcap E$	b:B, (b,d):t	$B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$, $B \sqsubseteq \forall t.D$	b:B, (b,d):s, (b,d):t, d:C, d:D
14.) $+D \sqsubseteq \exists r.D$	b:B, (b,d):t	$B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$, $B \sqsubseteq \forall t.D$, $D \sqsubseteq \exists r.D$	b:B, (b,d):s, (b,d):t, d:C, d:D, (d,e):r, e:D
15.) $d :!D \sqcup E$	b:B, (b,d):t, $d :!D \sqcup E$	$B \sqsubseteq \exists s.C$, $B \sqsubseteq \forall r.A$, $B \sqsubseteq \forall t.D$, $D \sqsubseteq \exists r.D$	b:B, (b,d):s, (b,d):t, d:C, d:D, (d,e):r, e:D, d:E

In Zeile 14 wird nun ein TBox-Axiom hinzugefügt, welches, falls es naiv aufgelöst werden würde, eine unendliche Schlussfolgerungskette erzeugt. Um dies zu verhindern, muss der verwendete Algorithmus dies erkennen und das weitere Auflösen von *e:D blocken*.

Anhand dieses Fallbeispieles lässt sich bereits eine zentrale Schwierigkeit feststellen. Diese ist das Unterscheiden von Fakten und abgeleitetem Wissen in der Materialisierung, sowie das Aufzeichnen der jeweiligen Herleitungen, die zu der Materialisierung führten. Ohne konkrete Informationen darüber, welche Teile der Materialisierung von bestimmten Fakten abhängen, kann ein Programm keine Löschoperationen durchführen, ohne das die komplette Materialisierung neu berechnet werden muss. Zu diesem Zweck wird der sogenannte *Herleitungsgraph* definiert, dessen Aufgabe es ist, die Herleitungsstruktur der Materialisierung darzustellen, um erweiterte Operationen auf der Materialisierung zu ermöglichen.

Definition 7. *Sei $G = \{V, E\}$ ein **Herleitungsgraph**, bestehend aus einer Menge Axiomknoten V und einer Menge gerichteter Herleitungskanten E die diese Knoten miteinander verknüpfen.*

Axiomknoten in V können sowohl TBox- als auch ABox-Axiome darstellen. TBox-Axiome in V besitzen die Form $S \sqsubseteq T$, wobei S und T jeweils Konzepte darstellen. Konzepte sind ein Term aus der Menge $\{A, !A, A \sqcap B, \exists r.A, \forall r.A, A \sqcup B\}$. A und B sind auch wiederum Konzepte, was die Syntax von Konzepten induktiv definiert.

ABox-Axiome sind zum einen Konzeptzuweisungen der Form x:X, was das Individuum x an Konzept X bindet. Zum anderen sind es Relationen der Form (a,b):r, welches die Individuen a und b in Relation r zueinander stellt.

Herleitungskanten in E bestehen aus einer Menge von Prämissen $P \subseteq V$ und eine Konklusion $K \in V$ in der Form eines Tupels (P, K). Jeder Axiomknoten k, der aus einer Menge anderer Axiome p hergeleitet wurde, ist mit diesen Knoten durch eine Herleitungskante (p,k) verbunden.

Die Schlussfolgerung ist transitiv, was im Falle des Löschens eines Axioms dazu führt, dass auch alle Nachfolger behandelt werden müssen. Jedes Axiom im Graphen muss entweder als Fakt markiert sein, oder über eine beliebig lange Schlussfolgerungskette vollständig aus Fakten hergeleitet sein. Jedes Axiom, welches dieses Kriterium nicht erfüllt, ist ungültig und muss aus der Materialisierung entfernt werden.

Durch die Nutzung einer solchen Graphenstruktur kann das Wissen, welches aus Axiomen entsteht, leicht zurückverfolgt werden und es lässt sich feststellen, mit welchen konkreten Schlussfolgerungen es hergeleitet wurde. Dies ermöglicht die Wartung der Daten sobald sich Fakten ändern, da sich die betroffenen Axiome im Graphen verfolgen lassen. Dieser Herleitungsgraph bietet die Grundidee für die erarbeitete Ontologieimplementation und wird in Abschnitt 3.3 genauer erläutert.

2.3 Related Work

Es wurden bereits Arbeiten zur inkrementellen Materialisierung von Ontologien durchgeführt. In den meisten Anwendungsfällen werden Materialisierungen in Datenbanken verwendet, die große Datensätze enthalten und diese, um die Verarbeitung zu erleichtern und zu optimieren, zerlegen und vorverarbeiten. Dies betrifft zum Beispiel SQL-Datenbanken

Staudt und Jarke liefern einen Ansatz zur inkrementellen Materialisierung im Hinblick auf eine Software, die direkten Zugriff auf die Ausgangsdaten hat, jedoch nicht

auf die Materialisierung. Dies soll primär in der kooperativen Nutzung von Serverdatenbanken Anwendung finden. Hier werden zwei Ansätze vorgestellt. Der erste Ansatz nimmt an, dass die Materialisierung sowohl Client-seitig als auch Server-seitig vorhanden ist, und beschreibt *Maintenance Rules*, die eine Menge von Änderungen errechnen und diese auf die Materialisierung anwenden. Der zweite Ansatz betrifft eine rein Client-seitige Materialisierung, die zusätzlich einen *Rederivation on Demand* verwendet. [SJ96]

Eine weitere verwandte Arbeit ist "Incrementally maintaining materializations of ontologies stored in logic databases" [VSM05]. In diesem Paper wird eine Vorgehensweise vorgestellt, die es ermöglicht, bei Änderungen in der Fakten- oder Regelmenge, Materialisierungen nicht auf naive Weise komplett neu zu berechnen, sondern nur das von der Änderung betroffene abgeleitete Wissen zu aktualisieren. Der Fokus von Volz, Staab und Motik liegt jedoch primär auf RDF/S und konkreten *Semantic Web* Anwendungen, wogegen die vorliegende Arbeit sich mit monotoner Beschreibungslogik allgemein auseinandersetzt.

Für die Faktenänderung wird die folgende Vorgehensweise vorgeschlagen. Dieser Algorithmus wird als *Delete and Re-Derive* (DRed) bezeichnet, wobei in [VSM05] eine deklarative Version genutzt wurde um die Kompatibilität mit logischen Datenbanken zu gewährleisten. Der Algorithmus wurde von [GMS93] vorgeschlagen und arbeitet wie folgt. Zuerst wird großzügig überschlagen welche Elemente gelöscht werden müssen (*overestimation of deletion*). Dies geschieht indem alle zu löschenden Elemente und ihre direkten Konklusionen markiert werden. Daraufhin werden alle zu löschenden Elemente geprüft. Jene Elemente, die durch andere Fakten im Programm hergeleitet werden können, werden von der Liste der zu löschenden Elemente genommen. Zuletzt werden die neuen Fakten und ihre Folgerungen eingefügt.

In der vorliegenden Arbeit wird eine ähnliche Vorgehensweise gewählt, um Axiome zu löschen. Dieser Löschalgorithmus wird in Kapitel 3.3.3 näher erläutert.

Furbach, Günther und Obermaier [FGO09] nutzen einen Ansatz zur Vorberechnung von generellen TBoxen in ALC, der beschreibungslogische Konzepte mit gerichteten Pfaden verknüpft. Dieses Vorgehen ermöglicht das Überprüfen von TBoxen auf Inkonsistenz und ein effizienteres Abfragen der TBox nach der Erstellung des Graphen.

Harrison und Dietrich [HD92] beschreiben eine inkrementelle Materialisierung, bei der ein *Abhängigkeitsgraph* (*dependency graph*) verschiedene Datensegmente verknüpft und bei Veränderungen der Ausgangsdaten feststellt, welche Segmente durch eine Änderung betroffen wurden und neu hergeleitet werden müssen. Änderungen in den Daten werden somit durch den Graphen propagiert und aktualisieren die Materialisierung.

Diese beiden Methoden, Wissen abzuleiten und in einem Graphen zu verknüpfen, ähneln dem in der vorliegenden Arbeit implementierten *Herleitungsgraphen*, besitzen jedoch andere Entwicklungsschwerpunkte und Implementationen.

3 Architektur und Implementation

Dieses Kapitel beschreibt die Architektur der einzelnen Komponenten des Prototypen aus abstrakter Sicht. Zuerst werden die benötigten Komponenten beschrieben. Danach wird ein Überblick über die statische Sicht geschaffen. In diesem Abschnitt werden die grundlegenden Datenstrukturen erläutert. Danach folgt eine Erläuterung der benutzten Schlussfolgerungsalgorithmen und auf welche Axiome sie anwendbar sind. Zuletzt wird der zur Laufzeit erzeugte Herleitungsgraph erläutert.

3.1 Ausgangssituation

Der ausgearbeitete Prototyp einer inkrementell materialisierenden Ontologie errechnet aus einer gegebenen Menge von Axiomen eine Materialisierung. Im Falle des Hinzufügens oder Löschens eines Axiomes kann durch die benutzte Herleitungsgraphenstruktur ein Großteil der Materialisierung bewahrt werden, was einen Laufzeitvorteil gegenüber einer naiven Implementierung erzeugt.

Ziel der Software ist es zunächst, eine Ontologie zu modellieren, die TBox- und ABox-Axiome enthält. Diese Axiome sollen die Ausdruckskraft von **ALC** beschreiben und es muss möglich sein, Wissen aus ihnen abzuleiten. Dieses Wissen ist wiederum in Axiomen formuliert.

Zu diesem Zweck muss zuerst eine Ontologiestruktur aufgebaut werden. Diese Ontologie muss eine Menge von ungeordneten, einzigartigen Axiomen enthalten können. Die einzelnen Axiome müssen jeweils Konzepte oder Individuen beschreiben und diese in Relation setzen. Aus einem oder mehreren Axiomen kann nun Wissen abgeleitet werden, welches direkt aus bestimmten Axiomen folgert.

Der Nutzer muss nun bestimmte Aktionen durchführen können. Dazu gehört das Eingeben von neuen Axiomen und das Löschen von existierenden Axiomen. Eine simple Textausgabe muss auch geliefert werden. Abbildung 1 zeigt ein Use-Case-Diagramm für die Benutzung der Software.

Die zwei Akteure die in dieser Software operieren sind der Nutzer, der Befehle eingibt, und die Ontologie, welche die Materialisierung verwaltet und die jeweiligen Materialisierungsalgorithmen ausführt. Der Nutzer hat drei grundlegende Operationen zur Auswahl. Diese sind der Befehl die Materialisierung auszugeben, ein Axiom hinzuzufügen und ein Axiom zu entfernen. Da Axiome jeweils ABox- oder TBox-Axiome darstellen können, wird das Hinzufügen und Entfernen von Axiomen als Spezialisierung der jeweiligen Aktion dargestellt. Die konkrete Implementation unterscheidet sich für die einzelne Spezialisierung, das System wählt jedoch automatisch den korrekten Algorithmus basierend auf dem eingegebenen Axiom.

Beim Hinzufügen von Axiomen zur Ontologie wird immer der Schlussfolgerungsalgorithmus aufgerufen, der das neue Axiom ableitet und das zusätzliche Wissen der Materialisierung hinzufügt. Analog dazu wird bei jedem Löschbefehl der Löschalgorithmus ausgeführt, der das abgeleitete Wissen entsprechend anpasst. Der genaue Ablauf dieser beiden Anwendungsfälle werden jeweils in Abschnitt 3.3.2 und 3.3.3 näher erläutert.

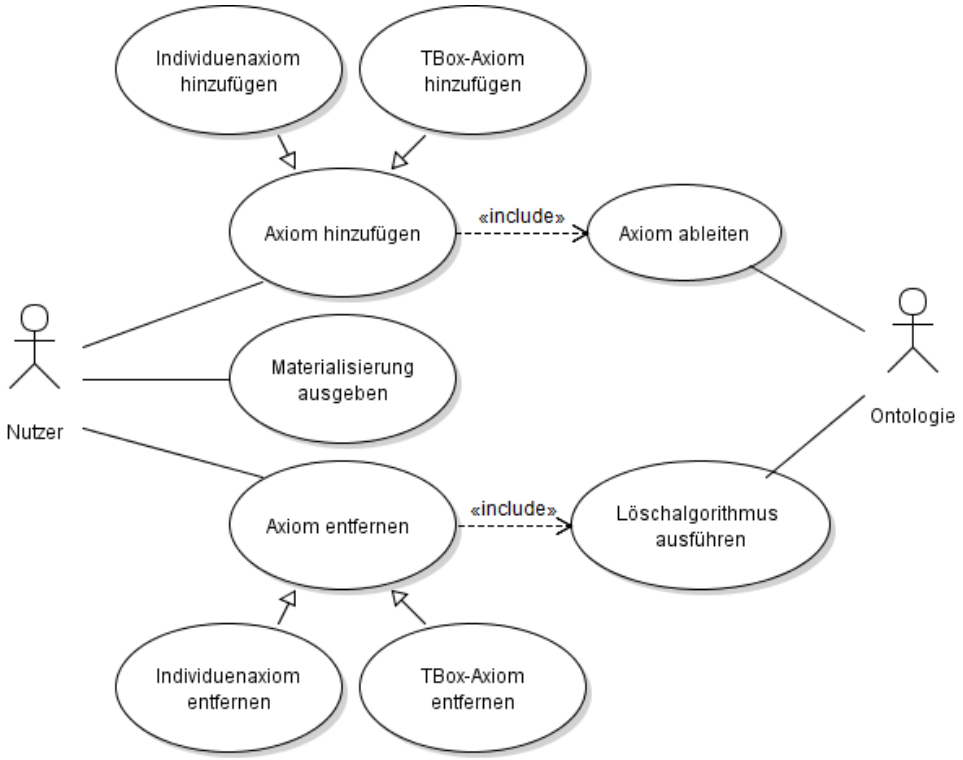

Abbildung 1. Use-Case Diagramm des Programmes

3.2 Statische Sicht

Dieses erste Unterkapitel beschreibt die statische Grundstruktur der Software. Dies beinhaltet die modellierten Datenstrukturen, sowie die Definition der Konzepte und den Aufbau der Ontologie mit den dazugehörigen Indizes.

Aus Gründen der Übersichtlichkeit wird die Architekturbeschreibung der statischen Sicht anhand der Klassendiagramme zunächst in drei Teile gespalten. Die erste Sektion befasst sich mit der Struktur der Axiomknoten. Danach wird die Struktur der Konzepte erläutert, die Syntaktische und Semantische Bedeutung der einzelnen Konzepte, so wie ihre induktive Definition. Danach folgt die Struktur der Ontologie.

3.2.1 Modellierung der Axiomknoten

Die abstrakte Klasse der *Axiomknoten* repräsentiert eine Datenstruktur, die ein einzelnes Axiom beinhaltet, enthält darüber hinaus jedoch einige Methoden und Attribute die für die spätere Materialisierung benötigt werden. In der Implementation und den dazugehörigen Klassendiagrammen wird diese Klasse aus Gründen der englischen Namenskonvention als *Axiomnode* bezeichnet.

Das Klassendiagramm in Abbildung 2 beschreibt die Klasse *Axiomnode*, sowie die von ihr abgeleiteten Klassen.

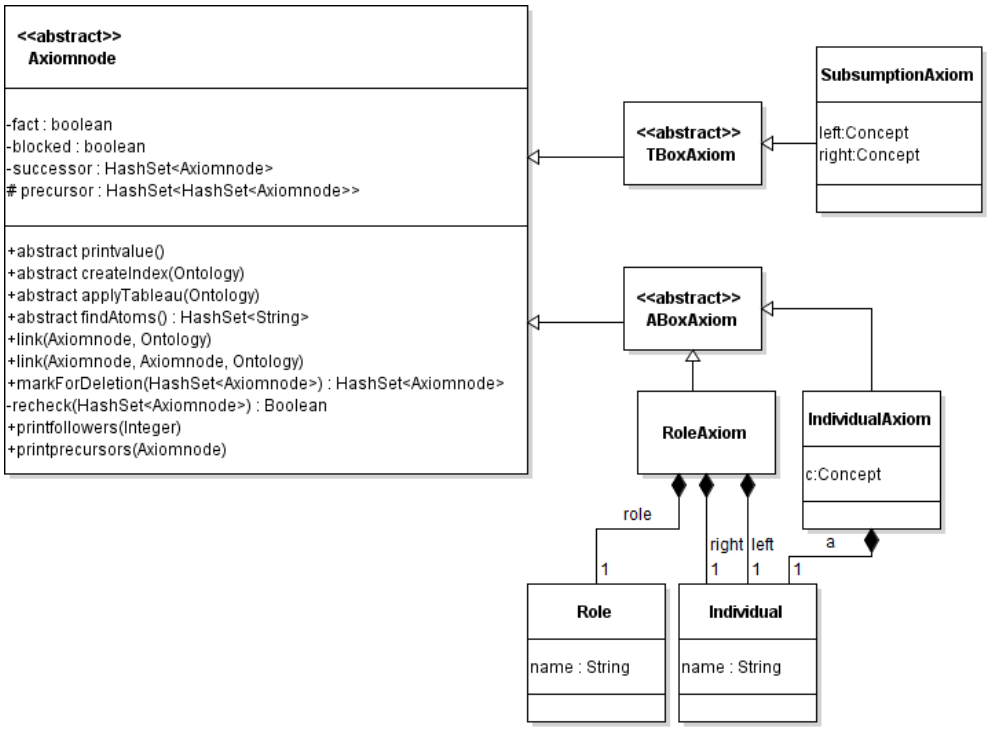

Abbildung 2. Klassendiagramm der Konzepte

Axiomnode Die abstrakte Klasse *Axiomnode* beschreibt die Superklasse aller Axiomknoten die in einer Ontologie erzeugt werden können. Jedes erstellte Axiom ist von dieser Klasse abgeleitet und implementiert alle ihre Methoden. Sie besitzt Attribute die in jedem Axiomknoten enthalten sein müssen. Diese Attribute umfassen die Information, ob der Knoten gegeben oder hergeleitet ist, welche Axiomknoten ihn herleiten und welche Axiomknoten aus ihm hergeleitet werden. Diese Informationen werden benötigt um später den Herleitungsgraphen zu konstruieren. Der konkrete Inhalt, das heißt das modellierte Axiom und die referenzierten Konzepte können erst in den entsprechenden Unterklassen eingefügt werden, da die benötigten Attribute sich für die verschiedenen Axiomtypen unterscheiden.

Axiome können in zwei verschiedenen Varianten auftauchen, TBox-Axiome und ABox-Axiome. Um diese Unterscheidung strukturell deutlich zu machen, bilden die abstrakten Klassen *TBoxAxiom* und *ABoxAxiom* die unmittelbaren Unterklassen von Axiomnode. Diese Unterscheidung hat zwar auf die momentanen Programmstruktur keinen Einfluss, wird jedoch aus Gründen der Erweiterbarkeit bezüglich neuer TBox- und ABox-Axiome beibehalten. Sie kann gegebenenfalls genutzt werden um den Prototypen mit neuen Funktionalitäten oder verschiedenen Arten von Axiomen zu erweitern.

Die Klassen *IndividualAxiom* und *RoleAxiom* sind konkrete Vertreter der ABox-Axiome. Sie modellieren jeweils die Zugehörigkeit eines Individuums zu einem Konzept, oder eine Rollenbeziehung zwischen zwei Individuen.

IndividualAxiom Ein *IndividualAxiom* besteht aus genau einem *Individual* a und einem *Concept* c, und beschreibt ein Axiom der Form $a : C$. Ein *IndividualAxiom* muss stets ein Individuum und ein Konzept besitzen, und diese Attribute können nicht verändert werden. Semantisch beschreibt dieses Axiom dass $a^I \subseteq C^I$, was letztendlich bedeutet, dass Individuum a eine Instanz von Konzept C ist.

RoleAxiom Die Klasse *RoleAxiom* besteht aus zwei Individuen $left$ und $right$. Diese Unterscheidung der zwei Individuenattribute ist wichtig, da Relationen nicht zwangsläufig kommutativ sind. Die Bezeichner $left$ und $right$ wurden gewählt, da Relationen im Allgemeinen als Tupel (a, b) geschrieben werden. $left$ bezeichnet somit das linke Individuum, das heißt das Subjekt von dem die Relation ausgeht, und $right$ das rechte Individuum, welches in der Relation als Objekt gilt. Jedes *RoleAxiom* besitzt ausserdem noch einen Rollenbezeichner, im Klassendiagramm dargestellt als Instanz der Klasse *Role*. Semantisch beschreibt ein *RoleAxiom* nun eine Relation von $left$ nach $right$ mit dem Bezeichner *role*. Eine Erweiterung um Rolleneigenschaften, wie zum Beispiel Transitivität, ist an dieser Stelle möglich, überschreitet jedoch die Größenordnung dieser Arbeit.

SubsumptionAxiom Die Klasse *SubsumptionAxiom* beschreibt die Struktur eines TBox-Axioms, welches den Subsumptionsoperator $\sqsubseteq$ enthält. Das Axiom besteht aus zwei Konzepten, die durch einen Subsumptionsoperator verbunden sind. Wie schon bei den Individuen einer Rolle, ist auch hier keine Kommutativität gegeben, sodass $left$ und $right$ unterschieden werden muss. Syntaktisch entspricht eine Instanz dieser Klasse einem Axiom $C \sqsubseteq D$. Aus semantischer Sicht modelliert dieses Axiom nun folgenden Zusammenhang. Seien C und D Konzepte, so ist $C \sqsubseteq D$ ein Subsumptionsaxiom und somit gilt $C^I \subseteq D^I$. Natürlichsprachlich formuliert bedeutet dies, dass alle Individuen aus Konzept C auch in Konzept D enthalten sind.

3.2.2 Modellierung der Konzepte

In diesem Abschnitt wird erläutert, wie die Klasse *Concept* strukturiert ist, welche Unterklassen sie besitzt und welche Bedeutung die einzelnen Unterklassen für den Aufbau von Konzepten besitzen.

Abbildung 3 zeigt einen Ausschnitt des Klassendiagrammes. Es beschreibt die Struktur der Klassen im Paket Concept.

Concept Das Interface *Concept* repräsentiert die Oberklasse aller Konzepte. Jedes mögliche Konzept ist von diesem Interface abgeleitet. Es enthält abstrakte Methoden die in allen Arten von Konzepten unterschiedlich implementiert werden. Die einzelnen Unterkonzepte sind induktiv strukturiert, so dass sie je nach Bedarf aus weiteren Konzepten bestehen können. So lassen sich Konzepte beliebig in einer Baumstruktur verknüpfen. Die Art der Verknüpfung wird im Folgenden bei den Beschreibungen der einzelnen Konzeptklassen näher erläutert.

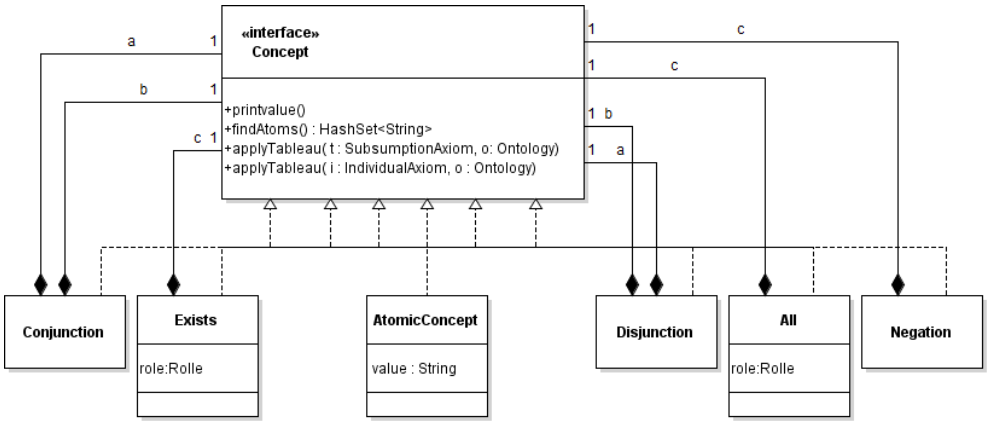

Abbildung 3. Klassendiagramm der Konzepte

Atomic Concept Das *Atomic Concept* steht für ein einzelnes atomares Konzept. Es besitzt nur einen Namen als Attribut und stellen in einer Konzeptverkettung einen Endzweig dar, der sich nicht weiter fortsetzt. Atomare Konzepte sind äquivalent, wenn sie denselben Namen besitzen.

Conjunction Die Klasse *Conjunction*, benannt nach dem englischen Wort für die logische Konjunktion, beschreibt den $\sqcap$-Operator. Analog zur Mengenlehre enthält ein Konzept, welches aus der Schnittmenge zweier Konzepte besteht, alle Individuen, die in mindestens einem der beiden Konzepte vertreten sind. Ein Individuum, welches in der Konjunktion zweier Konzepte enthalten ist, ist sowohl in dem einen, als auch dem anderen Konzept enthalten. Die zwei Konzepte, deren Schnittmenge gebildet werden soll, sind nur allgemein als *Concept* definiert. Dies bedeutet, dass jede Seite der Verknüpfung wiederum verschachtelte Konzepte enthalten kann. Um zwei Instanzen von *Conjunction* auf Äquivalenz zu testen, müssen jeweils die zwei Attribute Konzept a und Konzept b verglichen werden. Hierbei ist zu beachten, dass die Attribute kommutativ sind.

Exists Die Klasse *Exists* repräsentiert den Existenzquantor ($\exists$). Wie schon bereits in vorigen Kapiteln erläutert wurde, stellt der Existenzquantor sicher, dass eine Relation zu einem bestimmten Konzept besteht. Zu diesem Zweck besitzt die Klasse die Attribute *role:Rolle* und *c:Concept*. Auch hier bestehen keinerlei Einschränkungen bezüglich der Klasse von c, solange sie von *Concept* abgeleitet wurde. Die Klasse *Role* wurde bereits im vorigen Kapitel beschrieben.

All Der Allquantor($\forall$) besitzt, analog zum Existenzquantor, ein Rollenattribut und ein Konzeptattribut. Sie sind somit syntaktisch vergleichbar, stellen jedoch verschiedene Semantik dar und enthalten andere Schlussfolgealgorithmen.

Disjunction Der Disjunktionsoperator ($\sqcup$) ist wiederum syntaktisch sehr ähnlich zur Konjunktion. Er besitzt zwei Konzepte als Attribut welche kommutativ behandelt werden, modelliert jedoch einen anderen logischen Zusammenhang.

Negation Die Negation($\neg$) beschreibt das Inverse eines Konzeptes. Syntaktisch gesehen enthält diese Klasse nur ein Attribut der Klasse Konzept. Dieses Konzept kann eine Instanz von jedem Konzeptoperator sein. Es ist allerdings interessant zu bemerken, dass die Negation die einzige Klasse ist, bei der die Klasse des Attributes beim Schlussfolgerungsalgorithmus ins Gewicht fällt. Der verwendete Algorithmus wird später im Detail erläutert.

EL Die Klassen *Conjunction* und *Exists* modellieren jeweils die **EL** Konzeptoperatoren $\cap$ und $\exists$. Zusammen mit den atomaren Konzepten können sie genutzt werden, um jede EL zu beschreiben.

ALC Die Konzeptklassen *All*, *Disjunction* und *Negation* beschreiben jeweils den Allquantor ($\forall$), die Vereinigung ($\sqcup$) und die Negation ($\neg$). Durch das Hinzufügen dieser drei Operatoren zu einer EL lassen sich nun ALC Konzepte darstellen.

3.2.3 Modellierung der Ontologie

Dieser Abschnitt befasst sich mit dem initialen Aufbau der zu erstellenden Ontologie. Das folgende Klassendiagramm 4 zeigt den Aufbau der Ontologie.

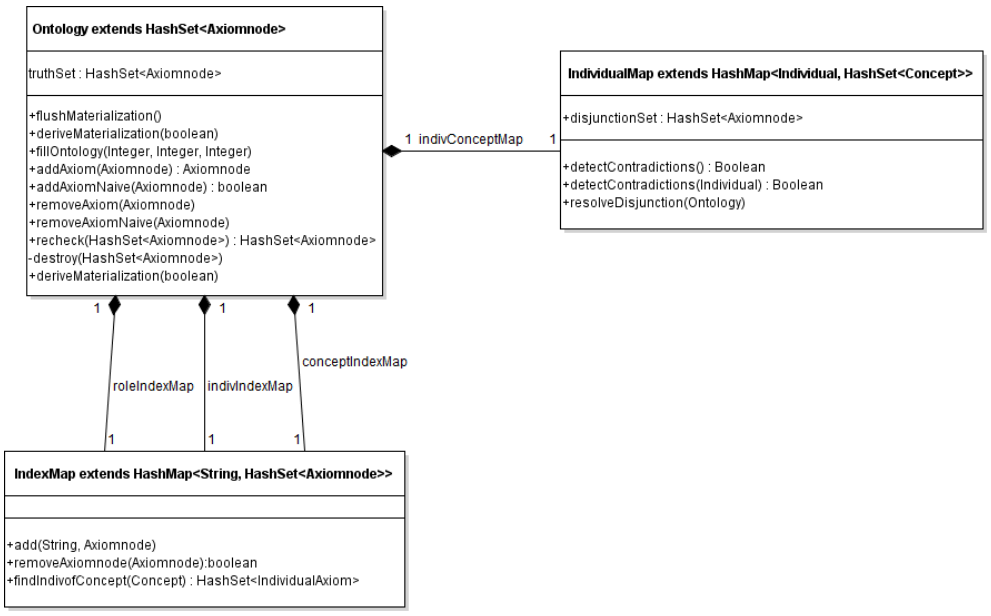

Abbildung 4. Klassendiagramm der Ontologie

Die Klasse Ontologie ist von einem HashSet von *Axiomnodes* abgeleitet. Sie enthält somit eine Menge von einzigartigen Axiomen. Dies sind alle Axiome die in der Ontologie enthalten sind, unabhängig davon ob sie hergeleitet oder gegeben sind.

Das *truthSet* enthält alle Axiome, die als Fakten gegeben sind. Rein semantisch ist dieses Set eine Teilmenge aller Axiome und enthält redundante Informationen, sie ist jedoch als separate Menge modelliert um einige Operationen zu erleichtern.

Die Ontologie besitzt mehrere *Indizes* die den Zugriff auf bestimmte Axiome erleichtern soll. Diese Indizes sind alle jeweils Instanzen der Klasse *IndexMap*. Diese IndexMaps verknüpfen jeweils einen simplen Bezeichner mit jedem Axiomknoten, in dem dieser Bezeichner auftaucht. Es gibt einen Index für Konzeptbezeichner, für Individuenbezeichner und für Rollenbezeichner. Sei zum Beispiel ein Individuenaxiom $a : C$ in der Ontologie enthalten. Dann existiert ein Eintrag in *indivIndexMap*, welcher das Axiom $a : C$ dem Schlüssel a zuordnet.

Darüber hinaus enthält die Ontologie auch eine sogenannte *individualConceptMap*. Sie ist eine Instanz der Klasse *IndividualMap* und wird benutzt um das Auflösen von Disjunktionen zu ermöglichen. Zu diesem Zweck beinhaltet sie Informationen über jedes Individuum und die jeweiligen atomaren Konzepte und negierten Konzepte die es repräsentiert. Sei a ein Individuum aus der Ontologie und $a : C,\ a : D,\ a : \neg E$ jeweils Axiome aus der Ontologie, so enthält die *individualConceptMap* einen Eintrag $\{C, D, \neg E\}$ der dem Schlüssel a zugeordnet ist.

3.3 Ontologie

Dieser Abschnitt beschreibt die Struktur der Ontologie und die darin erzeugten Herleitungsketten. Er beschreibt wie die einzelnen Axiome zur Laufzeit miteinander verknüpft werden, welche Muster sich ergeben und welche Operationen angewendet werden können. Zu den anwendbaren Operationen gehört das Hinzufügen und Löschen von Axiomen. Diese beiden Algorithmen werden im Detail beschrieben.

3.3.1 Herleitungsgraph

Der Herleitungsgraph ist die Struktur die entsteht, wenn zur Laufzeit Axiome in die Ontologie integriert werden und von anderen Axiomen als Vorgänger beziehungsweise Nachfolger eingetragen werden. Dadurch entsteht ein gerichteter Graph, in dem Knotenpunkte auf bestimmte Weise verbunden sind. Diese Struktur ist eine konkrete Implementation des in Definition 7 erläuterten abstrakten Herleitungsgraphen. Generell werden zwei oder drei Knoten in folgender Weise verknüpft. Seien A, B, C und D Axiomknoten.

C wurde aus A hergeleitet, so müssen A und C verknüpft werden. A wird nun in C als Vorgänger eingetragen und C wird in A als Nachfolger registriert. Somit lässt sich nachvollziehen welche Knoten voneinander abhängen, und in beide Richtungen navigieren.

D wurde durch Kombination von A und B hergeleitet, und müssen nun auch verknüpft werden. A und B sind somit beide notwendig um auf D zu schließen. Diese Bindung wird dargestellt, indem das Tupel (A, B) in D als Vorgänger eingetragen wird. In A und B wird D jeweils als einzelner Nachfolger gespeichert.

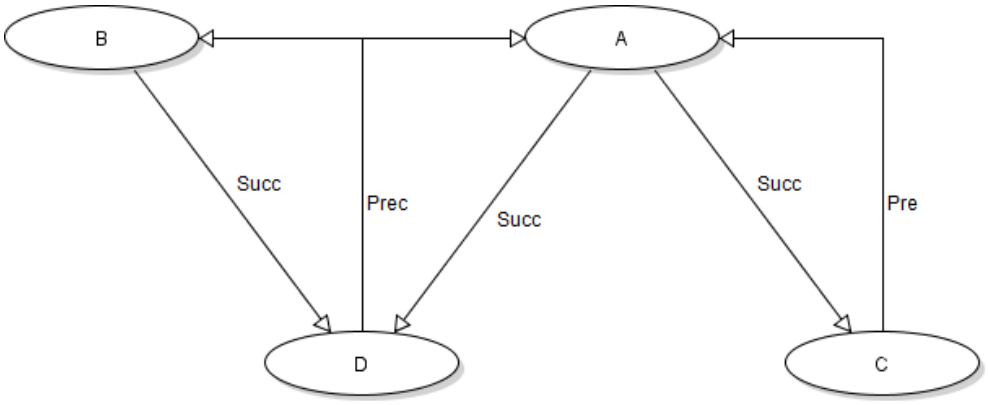

Abbildung 5. Beispiel verknüpfter Axiomknoten

Dieses Verknüpfen geschieht immer gleichzeitig in beide Richtungen sobald ein neues Axiom hergeleitet wurde, um sicherzustellen, dass es korrekt in den Herleitungsgraphen integriert wurde.

3.3.2 Schlussfolgerungsalgorithmus

Dieses Kapitel erläutert den benutzten Schlussfolgerungslgorithmus. Der Begriff *Schlussfolgerungsalgorithmus* bezeichnet hier den Algorithmus der benutzt wird, um neue Axiome aus existierendem Wissen für die Materialisierung herzuleiten. In diesem Abschnitt liegt der Fokus darauf, welche Klassen welchen Teil des Algorithmus implementieren, wie die Schlussfolgerung und das Herleiten von neuen Axiomen geregelt ist, und wie Spezialfälle behandelt werden.

Der Schlussfolgerungsalgorithmus kann sowohl auf ABox- als auch TBox-Axiome angewandt werden. Der konkrete Algorithmus ist in den jeweiligen Konzepten implementiert. In jeder Konzeptklasse wird allerdings unterschieden, ob der Algorithmus auf einem Individuenaxiom oder einem Subsumptionsaxiom ausgeführt wird. Allgemein gesehen erzeugt jeder Anwendungsschritt des Algorithmus neue Axiome. Ein erneutes Anwenden des Algorithmus auf bereits behandelte Axiomknoten hat keinen Effekt, stattdessen wird der Algorithmus auf die neu hergeleiteten Knoten angewandt bis keine weiteren Schlussfolgerungen mehr möglich sind. Die Rechenschritte, die in den jeweiligen Algorithmensegmenten durchgeführt werden, basieren auf den Expansion-Rules des Tableau-Algorithmus.

In Individuenaxiomen wird der Algorithmus auf das Konzeptattribut angewandt. Abhängig davon, zu welcher Konzeptklasse dieses Attribut gehört, wird daraufhin der entsprechende Algorithmus ausgeführt.

Im Falle eines Subsumptionsaxiomes wird der Algorithmus auf die rechte Seite des Axioms angewandt. Die linke Seite des Axioms bestimmt, auf welche ABox-Axiome die Regel angewandt werden kann. Die konkreten Schlussforlgerungsalgorithmen sind wiederum in den einzelnen Konzeptklassen enthalten.

Im Falle, dass der Algorithmus auf ein atomares Konzept trifft, wird es wie folgt behandelt. Ein Individuenaxiom der Form $a : C,\ Catomar$ wird gelesen. Zuerst wird das Konzept in der *individualConceptMap* eingefügt. Daraufhin werden alle TBox-Axiome nach möglichen Schlussfolgerungskandidaten durchsucht. Alle gefundenen Axiome besitzen die Form $C \sqsubseteq D$, wobei D ein beliebiges Konzept darstellt und C das Konzept des Individuenaxioms ist. Nun wird jeweils ein neues Individuenaxiom der Form $a : D$ erstellt und als Nachfolger an die verwendeten Axiome gebunden.

Analog dazu funktioniert die Schlussfolgerung mit einem Subsumptionsaxiom. Der Algorithmus erhält ein Axiom der Form $C \sqsubseteq D$. Ähnlich wie mit dem Individuenaxiom, wird an dieser Stelle die Ontologie nach Individuen der Form $a : C$ durchsucht und daraufhin für jedes gefundene Individuum ein neues Axiom der Form $a : D$ erstellt und mit seinen Vorgängern verknüpft.

Die allgemeine Umformungsregel für atomare Konzepte ist nun wie folgt:

$$\frac{a : C, C \sqsubseteq D}{a : D} \tag{14}$$

Wenn der Schlussfolgerungsalgorithmus auf eine Konjunktion trifft, wird wie folgt vorgegangen. Individuenaxiome der Form $a : C \sqcap D$ werden zuallererst auf den trivialen Fall geprüft, dass $C \equiv D$ ist. In diesem Fall folgt, dass $a : C$ als neues Axiom hinzugefügt werden kann. Ist dies nicht der Fall, so werden zwei neue Axiome erstellt, $a : C,\ a : D$ und jeweils als direkte Nachfolger verknüpft. Subsumptionsaxiome werden zusätzlich auf weitere triviale Fälle geprüft. Diese Fälle sind $C \sqsubseteq D \sqcap D \rightarrow C \sqsubseteq D$, $C \sqsubseteq C \sqcap D \rightarrow C \sqsubseteq D$ und $C \sqsubseteq D \sqcap C \rightarrow C \sqsubseteq D$. Dies wird getan, da ansonsten ein Axiom $C \sqsubseteq C$ entsteht, was eine Tautologie beschreibt und keinen logischen Wert besitzt. Trifft keiner der Sonderfälle zu, besitzt das Axiom die Form $C \sqsubseteq D \sqcap E$ und es werden zwei neue Axiome $C \sqsubseteq D$ und $C \sqsubseteq E$ hergeleitet.

Seien C,D und E unterschiedliche Konzepte, so gelten die zwei Herleitungsregeln:

$$\frac{a : (D \sqcap E)}{a : D, a : E} \tag{15}$$

$$\frac{C \sqsubseteq D \sqcap E}{C \sqsubseteq D, C \sqsubseteq E} \tag{16}$$

Ein Existenzquantor besitzt die folgenden Auflösungsalgorithmen. Wenn ein Subsumptionsaxiom in der rechten Seite einen Existenzquantoren als nächstes Konzept besitzt, es also die Form $C \sqsubseteq \exists r.D$ besitzt, wird analog zu atomaren Konzepten vorgegangen. Die Ontologie wird nach Axiomen der Form $a_1 : C ... a_n : C$ durchsucht und für jedes gefundene a_i ein neues Axiom $a_i : \exists r.D$ erstellt.

$$\frac{a : C, C \sqsubseteq \exists r.D}{a : \exists r.D} \tag{17}$$

Diese neu hergeleiteten Axiome werden daraufhin wiederum als Individuenaxiome mit Existenzquantorenkonzept behandelt und wie folgt abgeleitet. Zuerst wird geprüft ob eine Relation, wie sie von diesem Axiom verlangt wird, bereits existiert. Wenn dies zutrifft, kann sie mit diesem Axiom verknüpft werden und die Ableitung ist beendet. Ist dies nicht der Fall, wird aus dem gegebenen Axiom $a : \exists r.D$ zuerst geschlossen,

dass ein neues Individuum x in D existieren muss und erstellt das Axiom $x : D$. Daraufhin wird eine Relation $(a, x) : r$ generiert und die beiden neuen Axiome werden als Nachfolger von $a : \exists r.D$ deklariert.

$$\frac{a : \exists r.D}{x : D, (a, x) : r} \tag{18}$$

In der ersten Version der Schlussfolgerung, wurden sowohl in der TBox-Version, als auch der ABox-Version dieser Schlussfolgerung Individuen und Rollen hergeleitet. Dies führte zu aufwendigen und redundanten Suchalgorithmen. Die neue Methode um ein Subsumptionsaxiom nach Existenzquantoren aufzulösen nutzt kleinere Auflösungsschritte, ist jedoch dadurch simpler und modularer.

Das Entstehen endloser Auflösungsketten wird in der vorliegenden Arbeit verhindert, indem die Konzeptnamen lexikalisch verglichen werden. Ein zufälliges Subsumptionsaxiom wird so generiert, dass das Konzept auf der linken Seite immer das lexikalisch kleinste ist. So wird verhindert, dass zirkuläre Ableitungen entstehen.

Wie bereits erwähnt ähnelt der Allquantor ($\forall$) syntaktisch dem Existenzquantor. Dies bedeutet, dass der Algorithmus zum Ableiten von Subsumptionsaxiomen für Allquantoren identisch abläuft. Aus einem gegebenen Axiom $C \sqsubseteq \forall r.D$ und einer Menge $a_n : C$ entsteht $a_n : \forall r.D$.

$$\frac{a : C, C \sqsubseteq \forall r.D}{a : \forall r.D} \tag{19}$$

Die Verarbeitung für gegebene Individuenaxiome verläuft wie folgt. Zuerst sammelt der Algorithmus alle Relationen aus der Ontologie, für die gilt $(a, b_i) : r\ b_i \in I$. Für jede gefundene Relation wird nun ein neues Individuenaxiom der Form $b_i : D$ erstellt. Das eingegebene Individuenaxiom und die verwendete Rolle werden daraufhin zusammen als Vorgänger dieses neuen Individuums eingetragen.

$$\frac{a : \forall r.D, (a, x) : r}{x : D} \tag{20}$$

Wie auch im Existenzquantor nutzte die erste Version dieses Algorithmus größere Umformungsschritte. Dies führte dazu, dass für eingegebene Subumptionsaxiome eine Verknüpfungsmethode mit drei Vorgängerknoten benötigt wurde, die das TBox-Axiom, das Individuenaxiom, und die verwendete Rolle als Vorgänger einträgt. Diese Methode wird jedoch mit diesem Ansatz nicht mehr benötigt.

Die Negation wird zwar auch im Rahmen des Schlussfolgerungsalgorithmus aufgelöst, die verwendeten Rechenschritte entsprechen jedoch keiner Regel des Tableau-Algorithmus. Stattdessen sind es schrittweise Anwendungen der De Morganschen Gesetze, welche daraufhin ein weiteres Vorgehen mit den Tableau-Regeln ermöglichen. Sobald der Algorithmus auf eine Negation trifft, wird das negierte Konzept geprüft und abhängig von seiner Form die De Morganschen Gesetze angewandt.

Definition 8. *Die* ***De Morganschen Gesetze*** *[Goo07] sind definiert wie folgt:*
$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
$\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$
Und in erweiterter Form:
$\neg(\forall r.B) \Leftrightarrow \exists r.\neg(B)$
$\neg(\exists r.B) \Leftrightarrow \forall r.\neg(B)$

Hinzu kommt die triviale Umformung der zweifachen Negation$\neg\neg A \Leftrightarrow A$. Diese Gesetze lassen sich nun jeweils als Ableitungsregeln darstellen. Jedes Konzept, unabhängig davon ob es in TBox oder ABox-Axiomen auftaucht, kann nun in folgender Weise umgeformt werden.

$$\frac{\neg(\neg A)}{A}, \frac{\neg(A \sqcap B)}{\neg A \sqcup \neg B}, \frac{\neg(A \sqcup B)}{\neg A \sqcap \neg B}, \frac{\neg(\forall r.B)}{\exists r.\neg(B)}, \frac{\neg(\exists r.B)}{\forall r.\neg(B)} \tag{21}$$

Sei zum Beispiel das Axiom $C \sqsubseteq \neg(D \sqcup E)$ gegeben, so wird der Negationsoperator als oberster Operator erkannt und ein äquivalentes Axiom der Form $C \sqsubseteq \neg D \sqcap \neg E$ als Nachfolger hergeleitet. Mit diesem Schritt wurde zwar kein neues Wissen hergeleitet, da der verwendete Schlussfolgerungsalgorithmus jedoch nur sehr kleine Umformungsschritte betrachtet erfüllt das resultierende Konzept das Kriterium der Negationsnormalform, zumindest in dem für den Algorithmus relevanten Rahmen, und der Algorithmus kann einen nächsten Auflösungsschritt durchführen.

Im Falle der Disjunktion erfolgt die Behandlung von Subsumptionsaxiomen ähnlich wie bei Konjunktionen.

$$\frac{a : C, C \sqsubseteq D \sqcup E}{a : D \sqcup E} \tag{22}$$

Für die Schlussfolgerung aus Individuenaxiomen wurde der folgende Ansatz gewählt. Ein neues Individuenaxiom kann nur dann hergeleitet werden, wenn einer der zwei möglichen Pfade bereits einen Widerspruch erzeugen würde. Die Reihenfolge der Regelanwendungen spielt dabei keine Rolle, da eine Regel nur angewandt werden kann, wenn genau eine der beiden anwendbar ist.

$$\frac{a : D \sqcup E, a : \neg(D)}{a : E} \tag{23}$$

$$\frac{a : D \sqcup E, a : \neg(E)}{a : D} \tag{24}$$

An dieser Stelle ist außerdem zu beachten, dass beide möglichen Pfade auf Widersprüche geprüft werden müssen. Im Falle, dass beide alternativen Widersprüche erzeugen würden, kann keine Ableitung vorgenommen werden.

An dieser Stelle wird die bereits in 3.2.3 erwähnte *IndividualMap* verwendet, um zu prüfen, welche Individuen die nötigen Voraussetzungen erfüllen. Hier gibt es drei mögliche Resultate. Beide Pfade erzeugen einen Widerspruch, was bedeutet das Axiom kann nicht weiter aufgelöst werden. Nur einer der beiden Pfade kann widerlegt werden, was bedeutet der andere Pfad muss wahr sein, was in der Anwendung der jeweiligen Regel resultiert. Der dritte Fall besteht, wenn keiner der beiden Pfade widerlegt werden kann. Sobald eine Disjunktion nicht eindeutig im Schlussfolgerungsalgorithmus

aufgelöst werden kann, wird die entsprechende *Axiomnode* im *disjunctionSet* gespeichert und zur späteren Überprüfung aufbewahrt. Nach jeder Regeländerung muss nun geprüft werden, ob eins der ungelösten Disjunktionsaxiome nun eine Lösung besitzt. Da eine komplette Neuüberprüfung an dieser Stelle sehr aufwendig sein kann, wurde für den inkrementellen Algorithmus eine weitere Optimierung hinzugefügt. Nachdem ein neues Axiom der Ontologie hinzugefügt wurde, wird dieses Axiom auch der IndividualMap übergeben. Es werden nun alle ungelösten Axiome mit dem neuen Axiom verglichen und nur geprüft, wenn sie mindestens ein Literal, das heißt Individuen- oder Konzeptname, mit dem neuen Axiom gemeinsam haben. Somit können einige Überprüfungen eingespart werden.

3.3.3 Löschalgorithmus

Im Folgenden wird der Algorithmus beschrieben, der ein Axiom aus der Ontologie löscht. Da es das Ziel der inkrementellen Materialisierung ist, die Integrität der Ontologie zu bewahren, kann nicht einfach naiv ein Knoten gelöscht werden. Zum einen, weil aus diesem Knoten weitere Axiome folgen können, die, sollte der Knoten verschwinden, ihre Prämisse verlieren und somit inkorrektes oder nicht-verifizierbares Wissen darstellen. Zum anderen können die zu löschenden Knoten aus anderen Axiomknoten hergeleitet sein. Das Löschen eines solchen abgeleiteten Knoten würde somit zu einem Verlust des abgeleiteten Wissens führen. Dieses Wissen lässt sich jedoch sofort wieder herleiten, was das Ausführen des Löschalgorithmus überflüssig macht. Diese zwei Punkte bedeuten, dass der Löschalgorithmus nur auf Axiome angewandt werden kann, die als Fakten gegeben sind, das heißt, nicht nur als Herleitung aus anderen Axiomen existieren. Des Weiteren bedeutet es, dass zusätzlich zu dem ersten Knoten alle Knoten gelöscht werden müssen, die aus diesem Knoten folgern und keine alternative Herleitung besitzen.

Der konkrete Löschalgorithmus an sich läuft in mehreren Phasen ab. Die erste Phase markiert den zu löschenden Knoten als nicht faktisch und speichert seine Vorgänger. Die zweite Phase läuft rekursiv über alle Nachfolgeknoten und markiert sie als Löschkandidat. Dieser Markierungsalgorithmus arbeitet sehr großzügig, dass heißt er überschätzt die Löschkandidaten und erfasst alle Nachfolger, unabhängig von alternativen Vorgängern. Dies wird benötigt um zu vermeiden, dass Knoten als unmarkierte Vorgänger gezählt werden und das Löschen verhindern, zu einem späteren Zeitpunkt jedoch auch als Löschkandidat eingetragen werden. Die Methode, die diese Phase implementiert, ist *markForDeletion(HashSet<Axiomnode>)*. Sobald nun die Menge an Löschkandidaten zusammengetragen wurde, wird noch einmal geprüft, ob diese Kandidaten alternative Herleitungen besitzen die ihre Existenz rechtfertigen. Ist dies der Fall werden sie von der Liste der Löschkandidaten entfernt, andernfalls bleiben sie auf der Liste. Daraufhin wird der Algorithmus auf jeden Nachfolgeknoten ausgeführt, der auch als Löschkandidat gilt. *recheck(HashSet<Axiomnode>)* ist die Methode die diesen Teil implementiert. Die letzte Phase umfasst das *Zerstören* des jeweiligen Axioms. Diese Phase wird für jeden Löschkandidaten durchgeführt und beinhaltet die nötigen Operationen um alle Einträge zu entfernen die dieses Axiom beinhalten. Dies umfasst sowohl die diversen Indexeinträge in der Ontologie, als auch die Vorgänger- und Nachfolgereinträge in den übrigen Axiomknoten.

3.4 Prototypsoftware

Der implementierte Prototyp kann TBox und ABox-Axiome als Konsoleneingabe lesen und Axiome zu einer Ontologie sowohl hinzufügen als auch existierende Fakten entfernen. Wird dem eingegebenen Axiom ein „+“ vorangestellt, wird dieses Axiom der Ontologie hinzugefügt, falls es nicht schon bereits enthalten ist. Genauso führt ein „-“ vor dem Axiom dazu dass das Axiom gelöscht wird, sofern es in der Ontologie enthalten ist.

Ein Konzept C hat die Syntax { $D, !D, (D\&E), (D|E),\ \in\!r.D, \$r.D$ } wobei D und E beliebige Konzepte sind, und r ein Rollenname ist. Seien C und D beliebige Konzepte und a ein Individuum, so besitzen Individuenaxiome die Form *a:C* und Subsumptionsaxiome die Form $C <= D$.

Der auf dem Tableau-Algorithmus basierende Materialisierungsrechner unterstützt alle ALC-Befehle. Es gibt Methoden die zufällige ALC und EL TBox-Axiome generieren können. Diese könne zwar eine maximale Schachtelungstiefe besitzen, jedoch ist es nicht explizit möglich, beim Generieren von ALC-Axiomen das Entstehen von Widersprüchen in der Ontologie zu verhindern. Da der Schlussfolgerungsalgorithmus erst zur Laufzeit ausgeführt wird und der Algorithmus einen Small-Step Ansatz verfolgt, lässt sich beim Generieren eines TBox-Axioms weder eine Aussage über die enthaltenen atomaren Konzepte bezüglich ihrer Negation treffen, noch in welcher Weise diese mit bereits vorhandenen Individuen interagieren. Aus diesem Grund wurde das Vermeiden von Widersprüchen bei der Axiomgenerierung nicht behandelt. Für EL-Axiome gilt dies nicht, da durch das Fehlen des Negationoperators dort keine Widersprüche entstehen können.

Die Eingabe „naive“ wechselt zwischen der inkrementellen und der naiven Schlussfolgerungsmethode. Die Eingabe dieses Befehls hat keinen direkten Einfluss auf die bereits bestehende Ontologie, bestimmt allerdings die Behandlung von eingegebenen Befehlen. Im inkrementellen Modus werden alle Axiomänderungen mit dem inkrementellen Algorithmus ausgeführt, und im naiven Modus wird bei jedem Befehl die Materialisierung verworfen und hergeleitet.

Die Eingabe „print“ gibt den Inhalt der gesamten Ontologie als lesbaren Text in die Konsole aus. Hier werden Faktenaxiome linksbündig aufgezeigt und ihre Nachfolger in den darunterliegenden Zeilen jeweils mit Pfeilen eingeschoben aufgezählt, sowie die Axiome, die zu diesem Axiom führen.

Der Parser an sich ist relativ naiv aufgebaut und besitzt nur eine explizite Fallunterscheidung für bestimmte Zeichenketten. Hier können leicht neue Befehle eingebaut werden, oder erweiterte Algorithmen welche die Eingabe flexibler machen.

4 Laufzeitanalyse

In diesem Kapitel erfolgt ein Vergleich der Laufzeiten eines naiven Materialisierungsalgorithmus und des inkrementellen Algorithmus. Hierbei werden die Unterschiede in der Laufzeit beim Hinzufügen und Löschen von Axiomen gemessen, abhängig von der Größe der Ontologie. Hierbei ist zu erwarten, dass die Rechenzeit beider Ansätze abhängig von der Größe der Ontologie ansteigt, die benötigte Rechenzeit pro Rechenschritt in einem naiven Algorithmus jedoch weitaus schneller wächst, während der inkrementelle Ansatz in geringerem Maße ansteigt.

4.1 Parameter

In der Laufzeitanalyse wird zwischen den Folgenden zwei Ansätzen unterschieden.

Der *naive* Ansatz erstellt die Materialisierung bei jeder einzelnen Axiomänderung. Dies bedeutet, dass alle Materialisierungsdaten in jedem Schritt gelöscht und neu hergeleitet werden. Bei diesem Ansatz ist es zu erwarten, dass die benötigte Rechenzeit pro Befehl, abhängig von der Größe der Ontologie, stark ansteigt.

Der *inkrementelle* Ansatz nutzt den Herleitungsgraphen, um die Struktur der Materialisierung zu bewahren. Bei jeder Aktion beachtet der Algorithmus das bereits hergeleitete Wissen und beeinträchtigt nur die Axiome die durch den Nutzerbefehl direkt beeinflusst werden. Die Laufzeit dieses Algorithmus wird zwar auch in Abhängigkeit der Größe der Ontologie ansteigen, jedoch voraussichtlich in geringerem Maße als der naive Ansatz.

Die folgenden Faktoren haben Einfluss auf die Laufzeit. Das **Verhältnis von ABox- zu TBox-Axiomen** hat Einfluss darauf, welche Schlussfolgerungen getroffen werden können und beeinflusst somit die Anzahl der Axiome in der Materialisierung. **Die maximale Schachtelungstiefe** von zufällig generierten Konzepten in TBox-Axiomen wirkt sich darauf aus wie oft ein Axiom abgeleitet werden kann. Die Schachtelungstiefe dient beim Generieren des Konzeptes als maximale Tiefe für jeden generierten Ast, wobei die Grenze nicht notwendigerweise erreicht werden muss. Die Länge der auftretenden *Literale*, das heißt die Länge der Namen von Konzepten, Rollen und Individuen, hat auch einen Einfluss auf die Laufzeit, da eine zu geringe Menge dazu führt, dass sehr viele Schlussfolgerungen getroffen werden können, wogegen eine zu große Menge an Literalen die Wahrscheinlichkeit kleiner werden lässt, dass Schlussfolgerungen entstehen.

Im Rahmen der Laufzeitanalyse können die folgenden Aktivitäten verglichen werden.

Hinzufügen von Axiomen Das Hinzufügen eines Axiomes führt dazu, dass ein neues Fakt in der Ontologie eingetragen wird und neue Schlussfolgerungen sowohl direkt aus dem Axiom als auch in Kombination mit anderen Axiomen entstehen. Wie bereits zu erkennen läuft das *Hinzufügen von Axiomen* im inkrementellen Ansatz, besonders bei größeren Ontologien, weitaus schneller als im naiven Ansatz. Es ist zu erwarten dass der inkrementelle Ansatz selbst im *worst-case* niemals die Laufzeit des naiven Algorithmus benötigt.

Entfernen von Axiomen Das *Entfernen von Axiomen* ist eine weitere Funktion deren Laufzeit überprüft werden kann. Hier sind bedeutendere Erkenntnisse zu erwarten da der naive Algorithmus, im Gegensatz zum inkrementellen Ansatz, nicht in der Lage ist, Axiome zu löschen und gleichzeitig die Konsistenz der Materialisierung zu bewahren. Somit muss an dieser Stelle die Materialisierung neu berechnet werden. Die benötigte Rechenzeit wird voraussichtlich direkt von der Größe der Ontologie abhängen. Der inkrementelle Ansatz verwendet einen Algorithmus, der über die beeinträchtigten Axiome traversiert und diese nach Bedarf aus der Materialisierung entfernt. Die Laufzeit dieses Algorithmus ist abhängig von der Anzahl der Herleitungen die aus dem zu löschenden Axiom folgern. Dies ist zwar auch abhängig von der Größe der Ontologie, da die Axiome, und dadurch auch die jeweils möglichen Schlussfolgerungen, allerdings zufällig generiert sind, ist eine größere Varianz in der Laufzeit zu erwarten.

4.2 Durchführung

Zur Zeitmessung wurde zuerst der Javabefehl *System.currentTimeMillis()* benutzt. Da die Genauigkeit der Zeitmessung jedoch nicht für sehr kurze Laufzeiten im Millisekundenbereich ausreichte, wurde die Laufzeit stattdessen mit dem Javabefehl *System.nanoTime()* ausgelesen und die Messdaten daraufhin in Millisekunden konvertiert.

Es wird eine Testumgebung verwendet, die zufällige Axiome generiert. Um vergleichbare Ergebnisse zu erhalten werden für diesen Test einige Parameter zur Generierung von Axiomen auf feste Werte gesetzt. Diese Parameter bleiben für alle durchgeführten Tests gleich, sodass die resultierenden Laufzeitdaten sich nur in dem jeweiligen Algorithmus unterscheiden. Die *Größe* der Ontologie bezieht sich auf die Anzahl der Fakten in dieser Ontologie und nicht auf die Anzahl der hergeleiteten Axiome.

Die Generierungsparameter werden wie folgt gewählt. Die Wahrscheinlichkeit der Art der Axiome wird in diesem Test auf $p_{Ai} = 0.45$, $p_{Ar} = 0.45$ und $p_T = 0.1$ gesetzt. Somit ist zu erwarten, dass die resultierende Ontologie ca. 10% TBox-Axiome und 90 % ABox-Axiome besitzt, und die ABox in zu gleichen Teilen aus Rollenaxiomen und Individuenaxiomen besteht. Die maximale Schachtelungstiefe für Konzepte wird auf 4 gesetzt und die Länge der generierten Atomnamen wird auf 2 festgelegt.

Zur Messung der Laufzeit wird auf einer, im vornherein zufällig mit den selben Parametern gefüllten, Ontologie gearbeitet. Dazu wird eine Ontologie entsprechend der festgelegten Parametern mit einer bestimmten Anzahl Fakten gefüllt und daraufhin die Materialisierung berechnet. Danach werden einzelne Axiome abwechselnd gelöscht und hinzugefügt, und für jeden Schritt die benötigte Laufzeit aufgezeichnet. Beim Hinzufügen wird ein zufälliges Axiom generiert und der Ontologie hinzugefügt. Beim Entfernen wird ein zufälliges Axiom aus der Faktenmenge gewählt und gelöscht. Auf diese Weise kann die Laufzeit zum Entfernen und Hinzufügen gleichzeitig aufgezeichnet werden, und die Ontologie behält während des Laufzeittests annähernd dieselbe Menge Faktenaxiome. Im inkrementellen Ansatz wird dies pro erstellte Ontologie genau 10.000-mal durchgeführt und der resultierende Datensatz aufgezeichnet. Da dies im naiven Ansatz erheblich höhere Laufzeiten benötigt, werden dort weniger

Wiederholungen durchgeführt. Auf diesen Daten kann nun die Varianz und Standartabweichung der Daten abhängig von der Menge der Faktenaxiome berechnet werden. Dieser Prozess wird nun wiederum auf fünf bis zehn, mit denselben Parametern generierten, Ontologien ausgeführt um einen Mittelwert, in Abhängigkeit der Anzahl der Fakten, zu erlangen. Dieser Test wird mit verschiedenen Ontologien der Größe n= $\{1000, 3000, 5000, 7000, 10000\}$ durchgeführt.

4.3 Auswertung

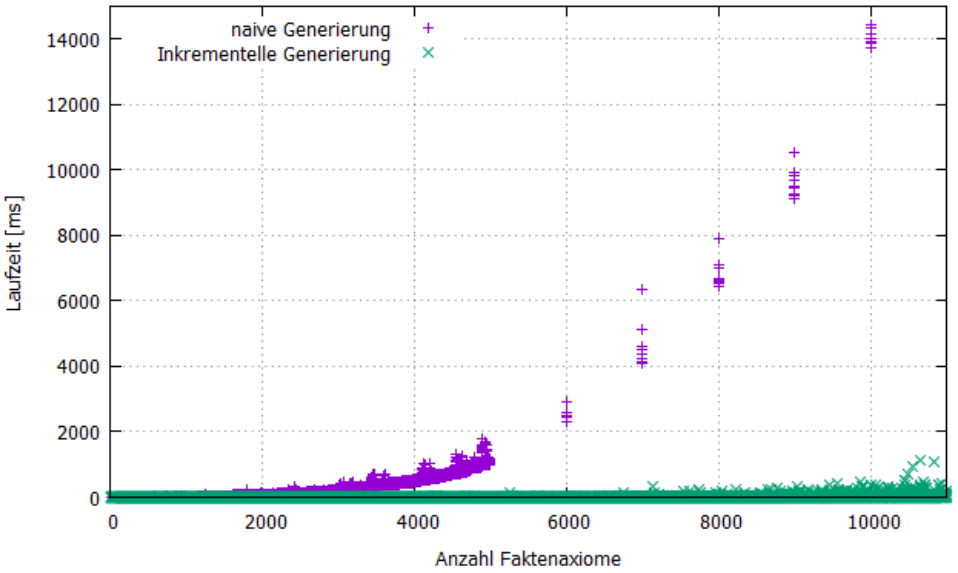

Abbildung 6. Stichprobe der Generierung einer Ontologie.

Die erste Beobachtung die sich machen lässt betrifft das Hinzufügen von Axiomen. Die Abbildung 6 zeigt die benötigte Rechenzeit für das Hinzufügen eines Axioms in Abhängigkeit der bereits vorhandenen Axiome in der Ontologie. Hier wurde jeweils ein Axiom zur Ontologie hinzugefügt und die Laufzeit gemessen. Der Datensatz *naive Generierung* stellt einen Durchlauf mit dem naiven Algorithmus dar. Hier wurde die Laufzeit pro Axiom stetig größer, was dazu führte, dass die Zeitmessung zu lange dauerte und auf Stichprobendaten zurückgegriffen werden musste. *inkrementelle Generierung* zeigt einen Graphen der durch den inkrementellen Algorithmus entstand.

Die beiden benutzten Datensätze sind aus zufällig generierten Axiomen gewonnen und nicht repräsentativ, es können allerdings schon einige offensichtliche Beobachtungen getroffen werden. Auf den ersten Blick ist zu erkennen, dass der inkrementelle Ansatz und der naive Ansatz grundlegend verschiedene Eigenschaften aufweisen. Beide Laufzeiten sind sich für kleine Ontologien sehr ähnlich, die Laufzeit des naiven Al-

gorithmus steigt jedoch sehr schnell mit der Anzahl der Faktenaxiome. Im Vergleich dazu bleibt der Mittelwert des inkrementellen Graphen relativ niedrig, es entstehen jedoch in größeren Ontologien einige Ausschlagswerte die vom Mittelwert abweichen, während ein Großteil der Messungen vergleichsweise niedrig liegen. Eine genauere Analyse der verschiedenen Messwerte folgt im nächsten Abschnitt.

Die folgenden zwei Tabelle basieren auf Messwerten, die jeweils mit einem *Lenovo Notebook N581* gemessen wurden.

Fakten	Axiome	naiv		inkrementell	
		Ø[ms]	σ	Ø[ms]	σ
1000	1262,5	19,08	6,068	0,1961	0.43788
3000	4818,995	300,723	38,99	0,4493	0,7423
5000	10203,9	1099,1	126,12063	1,7286	3,17
7000	18833,455	>4000,0	184	5,5139	12,13
10000	41234,75	>14000,0	∼800	27,4869	79,2534
12000	74211,498			100,37805	455,939

Tabelle 2. Laufzeitmessung: Hinzufügen von Axiomen

Die Tabelle 2 zeigt die mittlere Laufzeit für das Hinzufügen von Axiomen, jeweils im naiven und inkrementellen Ansatz für bestimmte Größen von Ontologien. Die Spalte *Fakten* enthält die Größe der Ontologie, und die Spalte *Axiome* die durchschnittliche Anzahl der Axiome die in der gesamten Materialisierung enthalten sind, das bedeutet die Summe der Faktenaxiome und der hergeleiteten Axiomen. Die restlichen Spalten zeigen jeweils die mittlere Rechenzeit pro Schritt in Millisekunden Ø[ms] und die durchschnittliche Standardabweichung σ.

Beim naiven Ansatz ist zu erkennen, dass die durchschnittliche Rechenzeit sehr schnell ansteigt, während die Standardabweichung vergleichsweise gering bleibt. Dies zeigt bereits das erste Problem mit der Erhebung von Testdaten im naiven Algorithmus. Ein einzelner, vollständiger Datensatz auf einer Ontologie mit 7000 Fakten, mit Messdaten für Hinzufügen und Entfernen, und 1000 Wiederholungen, benötigt bereits über 2 Stunden. Ein vergleichbarer Test im inkrementellen Algorithmus hingegen benötigt gerade einmal 6 Sekunden. Dies führt dazu, dass die Messwerte für den naiven Ansatz für größere Ontologien ungenauer werden, da sie sehr viel Zeit benötigen.

Auf der Seite des inkrementellen Ansatzes zeigt sich, dass die durchschnittliche Laufzeit durchgehend vergleichsweise niedrig bleibt, die Standardabweichung jedoch dagegen stark ansteigt. Dies ist darauf zurückzuführen, dass die benötigte Rechenzeit in diesem Ansatz nicht von der Größe der Ontologie n abhängt, sondern von der Anzahl der neuen Axiome, die aus dem hinzugefügten Axiom entstehen. Im Falle, dass ein Axiom eine große Menge von Folgerungen erlaubt, wird entsprechend mehr Zeit benötigt, was in vergleichsweise hohen Ausschlagswerten resultieren kann. Genauso kann der Fall auftreten, dass keine Folgerung getroffen werden kann, was zu einer sehr niedrigen Laufzeit führt. So kommt es zu einem geringen Durchschnittswert mit großer Standardabweichung.

Fakten	Axiome	naiv		inkrementell	
		Ø[ms]	σ	Ø[ms]	σ
1000	1262,5	18,1	5,7	0,2924	0,7449
3000	4818,995	295,916	36,4	0,3922	0,7330
5000	10203,9	1090,16	131,73	0,6879	1,5563
7000	18833,455	>4000	184	1,0686	3,2049
10000	41234,75	>14000	∼800	2,3430	16,3484
12000	74211,498			9,2822	192,225

Tabelle 3. Laufzeitmessung: Entfernen von Axiomen

Tabelle 3 zeigt die mittlere Laufzeit für das Entfernen von Axiomen, jeweils im naiven und inkrementellen Ansatz für bestimmte Größen von Ontologien.

Im naiven Algorithmus lässt sich erkennen, dass die Messwerte stark den Messwerten für das Hinzufügen ähneln. Dies ist darauf zurückzuführen, dass, wie bereits erwähnt, die Laufzeit im naiven Algorithmus nicht von dem hinzugefügtem oder entferntem Axiom abhängt, sondern durch das Neuberechnen der Ontologie entsteht. Dies bedeutet, dass sie effektiv dieselben Aktionen durchgeführt haben. Die Rechenzeit für das Herleiten einer Ontologie mit n Fakten unterscheidet sich, für große Mengen Faktenaxiome, nur unwesentlich von der Rechenzeit für $n+1$ Fakten. Beispielsweise benötigt die vollständige Herleitung einer Materialisierung mit n=8000 annähernd genauso lange wie dieselbe Ontologie mit n=8001. Da der naive Algorithmus lediglich die Faktenmenge um eins inkrementiert oder dekrementiert, und daraufhin die Materialisierung herleitet, sind die gemessenen Laufzeitwerte im naiven Ansatz für das Hinzufügen und Entfernen nahezu identisch.

Wie auch beim Hinzufügen ist der inkrementelle Algorithmus hier um einiges schneller als der naive Ansatz. Der Verlauf der Mittelwerte und Standardabweichung ist vergleichbar mit den Messwerten für das Hinzufügen. Die Mittelwerte verlaufen relativ gering, während die Standardabweichung stark zunimmt. Dieses Verhalten ist auf dieselben Gründe zurückzuführen wie die Werte des Hinzufügungsalgorithmus. Abhängig davon, wie viele Axiome von der Löschoperation betroffen sind, steigt die Rechenzeit, was zu einer großen Varianz in der Laufzeit führt.

Das Entfernen von Axiomen im inkrementellen Algorithmus steigt weniger stark mit der Größe der Ontologie als der Hinzufügungsalgorithmus. Dies ist dadurch zu erklären, dass beim Hinzufügen von Axiomen alle betroffenen und bisher ungelösten Disjunktionsaxiome geprüft werden. Da diese beim Löschalgorithmus bereits im Herleitungsgraphen enthalten sind, erzeugen sie keine zusätzliche Laufzeit und werden gegebenenfalls wie gewöhnliche Axiome gelöscht. So bildet sich ein ähnliches Laufzeitwachstum, welches sich in der Intensität unterscheidet. Die erhaltenen Ergebnisse lassen sich auf Graphen skizzieren.

Die zwei Abbildungen 7 und 8 bilden jeweils die gesammelten Daten aus Tabelle 2 und 3 dar, die mit einer Annäherungsfunktion verbunden wurden. Die Graphen wurden jeweils mit der Software GnuPlot [3] erstellt.

[3] http://www.gnuplot.info/

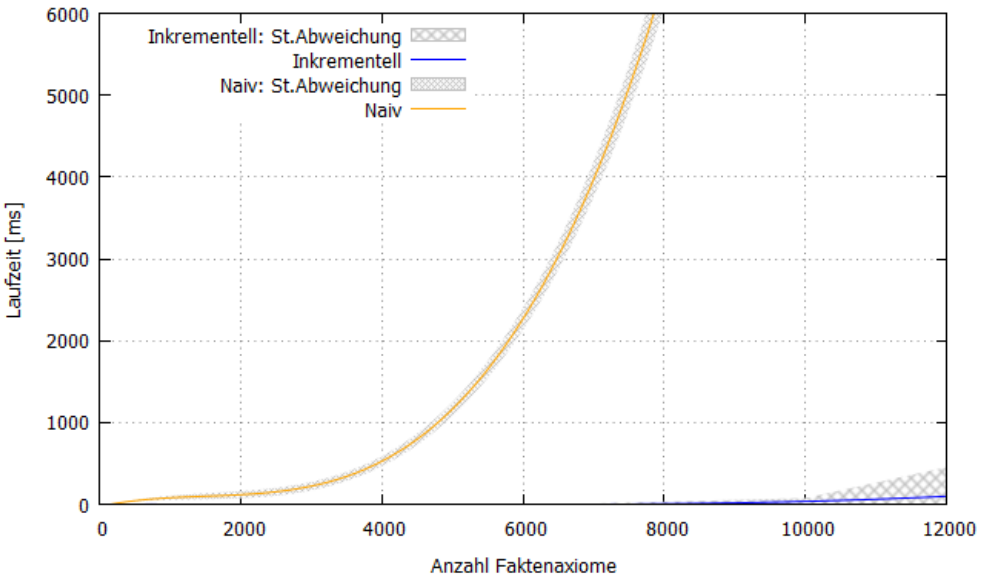

Abbildung 7. Annäherungsgraph zum Hinzufügen von Axiomen

Abbildung 7 skizziert die Laufzeit der beiden Ansätze im Hinblick auf das Hinzufügen von Axiomen. Beide Graphen zeigen die durchschnittliche Laufzeit des jeweiligen Algorithmus, sowie die Standardabweichung als graues Feld um den Graphen. Im naiven Algorithmus wird die Standardabweichung mit der Anzahl der Faktenaxiome zwar größer, bleibt jedoch im Vergleich zum generellen Verlauf des Graphen gering. Die Kurve des inkrementellen Algorithmus bleibt sehr flach, ab 10000 zeichnet sich jedoch, wie bereits in der Tabelle 2 erläutert, ein deutlicher Anstieg der Standardabweichung ab.

In Abbildung 8 zeigt sich der Unterschied zwischen den beiden Algorithmen deutlich. Dieser Graph skizziert die benötigte Laufzeit zum Entfernen von Axiomen in Abhängigkeit von der Anzahl der Faktenaxiome. Der Graph für den inkrementellen Algorithmus liegt in diesem Maßstab fast vollständig auf der x-Achse, während der Graph für den naiven Ansatz stetig ansteigt. Auch hier ist ein Anstieg der Standardabweichung in beiden Graphen zu erkennen.

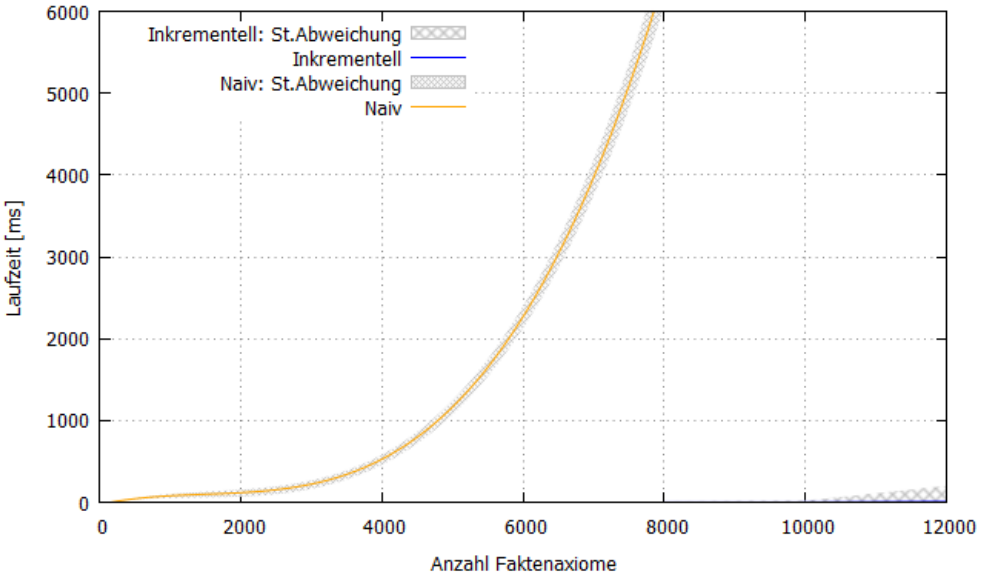

Abbildung 8. Annäherungsgraph zum Entfernen von Axiomen

Die gesammelten Daten legen den Schluss nahe, dass der inkrementelle Algorithmus niemals mehr Zeit benötigt als der naive Ansatz. Im Hinblick auf das Hinzufügen lässt sich dies leicht belegen. Beim Einfügen eines Axioms a in eine Ontologie mit n Fakten benötigt der inkrementelle Algorithmus lediglich die Zeit um a herzuleiten. Der naive Algorithmus hingegen behandelt n Axiome und Axiom a. Für eine Ontologie mit $n>0$ gilt somit immer, dass der naive Ansatz mehr Rechenzeit benötigt als der inkrementelle Algorithmus.

Die Algorithmen für das Löschen verlaufen im naiven und inkrementellen Ansatz relativ unterschiedlich. Der naive Löschalgorithmus führt die Herleitung für alle Faktenaxiome aus, unabhängig davon wie sie zusammenhängen. Dies bedeutet, dass er nahezu dieselbe Laufzeit besitzt, wie der naive Hinzufügungsalgorithmus. Der inkrementelle Löschalgorithmus traversiert alle Nachfolger des betroffenen Axioms, was nur die Nachfolger eines einzelnen Faktenaxioms betrifft. Dies bedeutet, dass nur die Axiome behandelt werden, die auch beim Hinzufügen dieses Axioms behandelt werden würden. Da sich aus den Tabellen 2 und 3 ablesen lässt, dass der inkrementelle Löschalgorithmus für Faktenmengen über 3000 schneller läuft als der inkrementelle Hinzufügungsalgorithmus, und der naive Ansatz für das Hinzufügen und Entfernen sehr ähnliche Laufzeiten benötigt, kann sich analog zum vorherigen Absatz schließen lassen, dass der inkrementelle Löschalgorithmus für größere Ontologien stets schneller ist als der naive Löschalgorithmus.

5 Ergebnis und Ausblick

In diesem Kapitel erfolgt die Zusammenfassung der erarbeiteten Erkenntnisse, eine kritische Diskussion der Resultate, sowie ein Ausblick auf mögliche Erweiterungen, die an dieser Arbeit vorgenommen werden können.

5.1 Zusammenfassung

Diese Arbeit stellt eine Software vor, die beschreibungslogische Axiome ableitet, eine Materialisierung erzeugt und diese inkrementell aktualisiert. Zuerst werden Grundlagen und Definitionen zu Beschreibungslogik erläutert, die in dieser Arbeit zum Einsatz kommen. Die Architektur und Implementation dieses Prototypen, und der verwendeten Datenstrukturen, werden erläutert und die Laufzeit daraufhin mit einem naiven Algorithmus verglichen. Der Quellcode der erarbeiteten Software liegt als CD bei, und wurde auf der Platform GitHub[4] hochgeladen, Die Laufzeitanalyse zeigt deutlich, dass der inkrementelle Algorithmus, der in dieser Arbeit implementiert wurde, für die gegebenen Parameter weitaus schneller läuft als ein naiver Ansatz.

5.2 Diskussion

Die Laufzeitauswertung zeigt, dass durch den inkrementellen Algorithmus Rechenzeit für das Warten der Materialisierung gespart werden kann. Die im Rahmen dieser Arbeit durchgeführte Laufzeitanalyse zeigt bereits große Unterschiede in der Laufzeit. Eine größere Menge an Messdaten, besonders für den naiven Algorithmus, erzeugt präzisere Daten, wurde jedoch aufgrund der Eindeutigkeit der Daten im Rahmen dieser Arbeit nicht näher betrachtet. Die vorliegenden Daten können den Verlauf skizzieren und durch den vergleichsweise großen Laufzeitunterschied der beiden Ansätze lassen sich eindeutige Schlüsse ziehen, jedoch werden für eine detailliertere Analyse mehr Datensätze benötigt. Hier bietet sich eine erweiterte Analyse an, welche die konkrete Laufzeit in Abhängigkeit weiterer Faktoren behandelt, wie zum Beispiel den Aufbau der ABox und TBox, Anzahl der hergeleiteten Axiome, sowie verschiedener Generierungsparameter. Eine empirische Signifikanzanalyse und Messwerte von verschiedenen Rechnern können dabei helfen, die Analyse objektiver werden zu lassen.

Der Prototyp verhindert bisher unendliche Schlussfolgerungsketten, indem der Axiomgenerator durch die Namensgebung der Konzepte dafür sorgt, dass keine zirkulären Schlussfolgerungen auftreten können. Dies wird erreicht, indem garantiert wird, dass jedes TBox-Axiom stets auf lexikalisch größere Konzeptnamen schließt. Dieser Ansatz verhindert zwar effektiv endlose Schlussfolgerungen, was für die Laufzeitanalyse ausreicht, ist jedoch nicht geeignet für die konkrete Verwendung in Datenbanken. Hier muss ein Blocking-System eingeführt werden und seperate Indizes für Konzeptnamen erstellt werden, um Fehler zu verhindern und dem Nutzer die Möglichkeit zu geben, Konzepte frei zu benennen.

[4] `https://github.com/bjoernkreutz/incremental_DL_materialization`

5.3 Ausblick

Diese Arbeit implementiert die grundlegenden Elemente einer **ALC**. Es ist allerdings denkbar, diese Arbeit auf neue Befehle der Beschreibungslogik zu erweitern. Wie bereits in 2.1.6 erwähnt, sind hier sind zum Beispiel Kardinalitätsrestriktion oder Rolleneigenschaften möglich. *Complexity of reasoning in Description Logics*[Zol13] bietet eine übersichtliche Liste von möglichen Erweiterungen die auf **ALC** angewandt werden können.

Der Parser, der die Nutzereingabe verwaltet, besitzt bisher nur eine simple Ausgabe der gesamten Ontologie. Es wäre denkbar, diesen Parser zu erweitern, um differenziertere Anfragen an die Ontologie zu stellen, wie zum Beispiel alle Konzepte eines Individuums, alle Individuen eines Konzeptes, oder sogar ein System von regulären Ausdrücken welches komplexere Anfragen an die Ontologie verarbeitet.

Ausserdem ist es möglich, die transitive Eigenschaft der Kanten im Herleitungsgraphen um einer Art Meta-Materialisierung zu erweitern, was womöglich die Laufzeit in besonders großen Materialisierungen beeinflussen kann.

Literatur

[BN03] Franz Baader und Werner Nutt. "Basic description logics." In: *Description logic handbook*. 2003, S. 43–95.

[BS01] Franz Baader und Ulrike Sattler. "An overview of tableau algorithms for description logics". In: *Studia Logica* 69.1 (2001), S. 5–40.

[FGO09] Ulrich Furbach, Heiko Günther und Claudia Obermaier. "A Knowledge Compilation Technique for ALC Tboxes." In: *FLAIRS Conference*. 2009.

[GM99] Ashish Gupta und Iderpal Singh Mumick. *Materialized views: techniques, implementations, and applications*. MIT press, 1999.

[GMS93] Ashish Gupta, Inderpal Singh Mumick und Venkatramanan Siva Subrahmanian. "Maintaining views incrementally". In: *ACM SIGMOD Record* 22.2 (1993), S. 157–166.

[Goo07] Reuben Louis Goodstein. *Boolean algebra*. Courier Corporation, 2007.

[HD92] John V Harrison und Suzanne W Dietrich. "Maintenance of Materialized Views in a Deductive Database: An Update Propagation Approach." In: *Workshop on Deductive Databases, JICSLP*. 1992, S. 56–65.

[KSH12] Markus Krötzsch, Frantisek Simancik und Ian Horrocks. "A description logic primer". In: *arXiv preprint arXiv:1201.4089* (2012).

[Lie10] Dr. Thorsten Liebig. *Wissensmodellierung und wissensbasierte Systeme*. `http://www.informatik.uni-ulm.de/ki/Edu/Vorlesungen/Wissensmodellierung/SS10/`. [Online; accessed 19-August-2015]. 2010.

[SJ96] Martin Staudt und Matthias Jarke. "Incremental maintenance of externally materialized views". In: *VLDB*. Bd. 96. Citeseer. 1996, S. 3–6.

[VSM05] Raphael Volz, Steffen Staab und Boris Motik. "Incrementally maintaining materializations of ontologies stored in logic databases". In: *Journal on Data Semantics II*. Springer, 2005, S. 1–34.

[Zol13] Evgeny Zolin. *Complexity of reasoning in Description Logics*. `http://www.cs.man.ac.uk/~ezolin/dl/`. [Online; accessed 19-July-2015]. 2013.