Universität Bremen

Fachbereich 3: Mathematik/Informatik

Bachelorarbeit

Subsumtionstest mittels SAT-Solver für eine leichtgewichtige Beschreibungslogik mit Funktionalität

Sascha Jongebloed
Martrikel-Nr.: 4004145

Erstgutachter: Prof. Dr. Thomas Schneider
Zweitgutachter: Prof. Dr. Dieter Hutter

14. April 2018

Abstract

Deutsch

In seiner Thesis "Complexity of Subsumtion in Extensions of $\mathcal{EL}$" hat Haase (2007) verschiedene Erweiterungen der Beschreibungslogik $\mathcal{EL}$ diskutiert und dabei unter anderem für die Erweiterung um globale Fuktionalität ($\mathcal{EL}^{\mathcal{F}}$) von Rollen gezeigt, dass Subsumtion bezüglich azyklischen TBoxen Co-NP-vollständig ist.

In dieser Arbeit wird das Subsumtionsproblem für $\mathcal{EL}^{\mathcal{F}}$ auf das Erfüllbarkeitsproblem der Aussagenlogik reduziert um so einen Reasoner zu implementieren, der den "schweren" Teil an einen SAT-Solver auslagert. Dabei wird in dieser Arbeit zunächst die Reduktion beschrieben. Danach wird auf die Implementierung des Reasoner eingegangen, der einzelne Subsumtionen mithilfe von SAT-Solvern entscheiden kann. Zum Schluss vergleichen wir den hier implementierten Reasoner mit bereits existierenden Tableau- und Konsequenz-basierten Reasonern.

Um den implementierten Reasoner zur vollständigen Klassifikation einer Ontologie in $\mathcal{EL}^{\mathcal{F}}$ nutzen zu können, werden weitere Optimierungen benötigt, die in dieser Thesis skizziert werden.

English

In his thesis "Complexity of Subsumtion in Extensions of $\mathcal{EL}$" Haase (2007) discussed various extensions of the description logic $\mathcal{EL}$. The thesis shows that subsumption is intractable for, amongst others, the extension by global functionality ($\mathcal{EL}^{\mathcal{F}}$) wrt. acyclic TBoxes.

This thesis shows, that it is possible to reduce subsumtion for $\mathcal{EL}^{\mathcal{F}}$, and to implement a reasoner to decide subsumption, using a SAT-Solver for the "hard" part of the decision. In the first part we describe the reduction. Afterwards we discuss the implementation of the reasoner, which can decide single subsumptions with the help of SAT-Solver. Finally we compare the runtime of our implemented Reasoner with Tableau-based and consequence-based Reasoner.

Further optimizations, that are described in this thesis, are needed to use the implemented reasoner for the classification of an ontology in $\mathcal{EL}^{\mathcal{F}}$.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Um Wissen darzustellen und zu modellieren haben sich in den letzten Jahrzehnten verschiedene Möglichkeiten herausgebildet. Ontologiesprachen wie OWL, auf der große Wissensdatenbanken, wie die medizinische Datenbank SNOMED, basieren, haben Beschreibungslogiken als Grundlage, die es erlauben Wissen in einer formalen logikbasierten Sprache zu repräsentieren.

Eine bekannte Beschreibungslogik ist dabei ALC. ALC nutzt die Basisoperatoren Konjunktion ($\sqcap$), Disjunktion ($\sqcup$), Negation ($\neg$), Existenzrestrektion für Rollen ($\exists r.C$) und Werterestrektion für Rollen ($\forall r.C$). Für diese sind aber relevante Entscheidungsprobleme nicht effizient entscheidbar. Ein Beispiel für ein solches Entscheidungsproblem ist das Subsumtionsproblem mit TBoxen, das danach fragt, ob für alle Interpretation $\mathcal{I}$ gilt, dass die Extension eines Konzeptes C eine Teilmenge der Extension eines anderen Konzeptes D ist (also gilt $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, wir schreiben dann: $C \sqsubseteq D$).

Um solche relevanten Probleme auch für sehr große Ontologien entscheiden zu können, haben sich leichtgewichtige Beschreibungslogiken herausgebildet. Eine dieser Logiken ist $\mathcal{EL}$, für die z.B. die Subsumtion mit generellen TBoxen in polynomieller Zeit entscheidbar ist (vgl. Haase, 2007, S.21). Dabei ist $\mathcal{EL}$ auf die Basisoperatoren Konjunktion ($\sqcap$), Existenzrestriktion für Rollen ($\exists r.$) und das Top - Konzept ($\top$) beschränkt.

In seiner Arbeit "Complexity of Subsumtion in Extensions of $\mathcal{EL}$" hat Haase (2007) dabei für verschiedene Erweiterungen von $\mathcal{EL}$, wie die Erweiterung um Rollenkonjunktion und -disjunktion und at-least Restriktionen für Rollen gezeigt, dass für Subsumtion für diese immer noch in polynomieller Zeit entscheibar ist (vgl. Haase, 2007, S.34). Für andere Erweiterung hat Haase in seiner Arbeit gezeigt, dass Subsumtion nicht mehr effizient entscheidbar ist. Beispielsweise kann man hier die Erweiterung von $\mathcal{EL}$ um globale Funktionalität ($\mathcal{EL}^{\mathcal{F}}$) mit azyklischen TBoxen nennen, bei der das Subsumtions-Problem co-NP-vollständig ist (vgl. Haase, 2007, S.57).

In dieser Arbeit soll ein Reasoner implementiert werden, der das Subsumtionsproblem für die die Logik $\mathcal{EL}^{\mathcal{F}}$ entscheiden kann. Dabei soll der "schwere" Teil des Problems auf das Erfüllbarkeitsproblem der Aussagenlogik reduziert werden, um optimierte SAT-Solver zu verhalten.

1.2 Leitfragen

In dieser Thesis behandeln wir die Frage, ob es möglich ist, für eine nicht-effiziente Erweiterung von leichtgewichtigen Beschreibungslogiken spezialisierte Reasoner zu entwickeln, die den "schweren" Teil an einen SAT-Solver auslagern und sich dank der Optimierungen im SAT-Solver relativ performant verhalten.

2 Beschreibungslogik

Die ersten Ansätze für Wissensrepräsentationssysteme entstanden in den 1970er Jahren. Diese teilten sich zuerst in Logik-basierte Systeme, und in nicht-Logik-basierte Systeme, die oft an kognitiven Modellen angelehnt waren, auf. Zu den nicht-Logik-basierten Systemen gehörten semantische Netzwerke und Frames, die in den 1960er (semantische Netzwerke nach Quillian 1967) bzw. den 1980er (Frames nach Minsky 1981) Jahren entwickelt wurden. Diese verfolgten den Ansatz, aus dem repräsentierten (expliziten) Wissen weiteres (implizites) Wissen abzuleiten.

Aufgrund der eher menschennahen Gestaltung von Systemen, wie semantischen Netzwerken und Frames, wurden diese oft als ansprechender und effektiver als Logik-basierte Systeme angesehen. Der Nachteil der semantischen Netzwerke und Frames war, dass es keine allgemeine formale Notation für die Semantik dieser Systeme gab. Dies änderte sich, nachdem Hayes (1979) zeigte, dass man die Semantik von Frames in First-Order-Logic darstellen kann. Weiterführend zeigten Levesque and Brachman (1985), dass semantische Netzwerke und Frames als Fragmente der First-Order-Logic angesehen werden konnten.

Aus diesen Entdeckungen entstanden die ersten Systeme, die auf Beschreibungslogiken basierten, z.B. KL-One von Brachman and Schmolze (1985). Zur Jahrtausendwende gewannen Beschreibunglogik als Grundlage des Semantic Web und insbesondere der Ontologiesprache OWL (Web Ontology Language), die 2004 vom World Wide Web Consortium entwickelt und empfohlen wurde (Patel-Schneider et al., 2004), an Bedeutung. Heute existieren OWL-Ontologien für verschiedene Bereiche, wie z.B. der Medizin (die medizinische Ontologie SNOMED).

Im nächsten Schritt werden wir die grundlegende Beschreibungslogik ALC (Attribute Language with Complement, Schmidt-Schauß and Smolka 1991) betrachten, deren Operatoren unter anderem auch in OWL realisiert sind.

2.1 Grundlagen der Beschreibungslogik

In diesem Kapitel wollen wir die Grundlagen der Beschreibungslogiken skizzieren, und die in der weiteren Arbeit verwendete Notation einführen. Diese enspricht der Notation aus Baader et al. (2008).

2.1.1 Konzepte

Konzeptnamen sind Klassen von Individuen. Die (potenziell unendliche) Menge von Konzeptnamen beschreiben wir mit N_C. *Rollennamen* beschreiben dabei Relationen zwischen den Konzepten. Die (ebenso potenziell unendliche) Menge von Rollennamen beschreiben wir mit N_R. Beispiele für ein Konzept und eine Rolle könnte z.B. “Mutter” und “hatKind” sein.

$\mathcal{ALC}$-Konzepte sind dabei nach Schmidt-Schauß and Smolka (1991, vgl. S.5) induktiv wie folgt definiert:

- Jeder Konzeptname $A \in N_C$, sowie $\top$ ist ein Konzepte
- Wenn C und D Konzepte, so auch $\neg C$ (Negation von C), $C \sqcap D$ (Konjunktion von

C und D) und $C \sqcup D$ (Disjunktion von C und D).

- Wenn C ein Konzept ist, und r eine Rollename, dann sind $\exists r.C$ (Existenzrestriktion) und $\forall r.C$ (Werterestriktion) Konzepte.

2.1.2 Interpretation

Nun betrachten wir *Interpretationen* um die gerade genannten Operatoren genauer zu definieren. Schmidt-Schauß and Smolka (1991, vgl. S.5) definiert eine Interpretation $\mathcal{I}$ als ein Paar $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. $\Delta^{\mathcal{I}}$ bezeichnet dabei die *Domäne* von I (eine nicht-leere Menge) und die Funktion $\cdot^{\mathcal{I}}$ bildet jeden Konzeptnamen in N_C auf eine Teilmenge von $\Delta^{\mathcal{I}}$, und jede Rolle in N_R auf eine eine Teilmenge von $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ ab, sodass für alle Konzepte C, D und Rollen die folgenden Gleichungen gelten:

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
- $(\forall r.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \forall (a,b) \in r^{\mathcal{I}} : b \in C^{\mathcal{I}}\}$
- $(\exists r.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \exists (a,b) \in r^{\mathcal{I}} : b \in C^{\mathcal{I}}\}$
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$

2.1.3 TBox

TBoxen können im Allgemeinen in azyklische und generelle TBoxen aufgeteilt werden. Generelle TBoxen sind dabei eine endliche Menge von Konzeptinklussionen der Form:

$$A \sqsubseteq B,$$

wobei A und B Konzepte sind. Wir kürzen $A \sqsubseteq B$ und $B \sqsubseteq A$ mit $A \equiv B$ ab. Eine azyklische TBox ist eine Menge von Konzeptinklusionen $A \sqcap B$ und Konzeptdefintionen $A \equiv B$, mit A Konzeptname (also $A \in N_C$). Außerdem kommen keine Zyklen vor, dass heißt, dass Konzepte nicht in ihrer eigenen Definition oder in Definitionen von Konzepten vorkommen, die indirekt von dem Konzept abhängig sind. Baader (2003, vgl. S.17-19)

Als Beispiel sei folgende TBox, hauptsächlich entnommen aus Baader (2003, S.56), gegeben:

$$\begin{aligned}
Woman &\equiv Person \sqcap Female \\
Man &\equiv Person \sqcap \neg Woman \\
Mother &\equiv Woman \sqcap \exists hasChild.Person \\
Father &\equiv Man \sqcap \exists hasChild.Person \\
Parent &\equiv Father \sqcup Mother \\
Grandmother &\equiv Mother \sqcap \exists hasChild.Parent \\
MotherWithoutDaughter &\equiv Mother \sqcap \forall hasChild.\neg Woman \\
Wife &\equiv Woman \sqcap \exists hasHusband.Man
\end{aligned} \tag{1}$$

Diese TBox ist offensichtlich azyklisch, da jedes Konzept nur eindeutig definiert wird, und keine Zyklen in den Definitionen vorkommen. Konzepte die nicht in der TBox definiert werden, nennen wir primitive Konzepte (bei der obigen TBox ist z.B Person ein primitives Konzept). Im Verlauf der Arbeit wollen wir uns aus später genannten Gründen auf azyklische TBoxen konzentrieren.

Eine TBox $\mathcal{T}$ *erfüllt* wird von einer einer Interpretation $\mathcal{I}$, wenn für jedes $A \equiv C \in \mathcal{T}$ gilt $A \equiv C \in \mathcal{T} gdw. A^{\mathcal{I}} = C^{\mathcal{I}}$ und für jedes $A \sqsubseteq C \in \mathcal{T}$ gilt $A \sqsubseteq C gdw. A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. Diese Interpretation nennen wir in diesem Fall ein Modell von $\mathcal{T}$ und schreiben $\mathcal{I} \models \mathcal{T}$.

2.1.4 Entscheidungsprobleme

In der Beschreibungslogik werden häufig (Baader, 2003, vgl. z.B.: S.66) folgende 4 Entscheidungprobleme betrachtet (sei $\mathcal{T}$ eine TBox):

- Erfüllbarkeit: Ein Konzept C ist bezüglich $\mathcal{T}$ erfüllbar, wenn ein Model $\mathcal{I}$ von $\mathcal{T}$ existiert, sodass das $C^{\mathcal{I}}$ nicht leer ist. Wir schreiben dann $\mathcal{T} \models C$.
- Subsumtion: Ein Konzept C wird bezüglich einer TBox $\mathcal{T}$ vom Konzept D subsumiert, wenn für jedes Model $\mathcal{I}$ von $\mathcal{T}$ gilt: $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Wir schreiben dann $\mathcal{T} \models C \sqsubseteq D$ oder $C \sqsubseteq_{\mathcal{T}} D$
- Äquivalenz: Zwei Konzepte C und D sind äquivalent, wenn für jedes Model $\mathcal{I}$ von $C^{\mathcal{T}}$ gilt: $C^{\mathcal{I}} = D^{\mathcal{I}}$. Wir schreiben dann $\mathcal{T} \models C \equiv D$ oder $C \equiv_{\mathcal{T}} D$.
- Zusammenhangslosigkeit: Zwei Konzepte C und D sind zusammenhangslos, wenn für jedes Model $\mathcal{I}$ von $C^{\mathcal{T}}$ gilt: $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$.

Da diese Entscheidungsprobleme jeweils in polynomieller Zeit wechselseitig aufeinander reduzierbar sind (Baader, 2003, vgl. S.67), wird oft häufig nur das Subsumtionsproblem betrachtet. Zudem bleibt anzumerken, dass wir die Entscheidung über die Subsumtion zwischen allen Konzepten (also $N_C \times N_C$) Klassifikation nennen.

2.1.5 Komplexität

Um die Komplexität der Entscheidungsprobleme in ALC zu betrachten, reicht es, hier nur das Erfüllbarkeitsproblem zu betrachten, da, wie bereits im vorherigen Kapitel erwähnt, die verschiedenen Probleme in polynomieller Zeit aufeinander reduzierbar sind.

Je nach TBox gibt es verschiedene Komplexitätschranken für das Erfüllbarkeitsproblem:

Tabelle 1: Komplexität mit verschiedenen TBoxen

Art der TBox	**Komplexität**	**Gezeigt in**
Leere TBox	PSpace-vollständig	Schmidt-Schauß and Smolka (1991, vgl. S.21,Theorem 6.3)
Azyklische TBox	PSpace-vollständig	Lutz (1999, vgl. Theorem 1)
Generelle TBox	ExpTime-vollständig	Baader (2003, vgl. S.126,Theorem 3.27)

Trotz dieser hohen unteren Schranken, schaffen spezialisierten Reasoner wie z.B.: Fact++ die Subsumtion auf relativ großen Ontologien in angemessener Zeit zu entscheiden. Für besonders große Ontologien kann aber eine geringere Komplexität notwendig werden. Zu diesem Zweck haben sich leichtgewichtige Beschreibungslogiken herausgebildet.

2.2 Leichtgewichtige Beschreibungslogiken

Leichtgewichtige Beschreibunglogiken erlauben nur eine begrenzte Auswahl an Basisoperatoren. Anders als bei ALC sind die oben genannten Entscheidungsprobleme auf diesen Logiken aber effizient entscheidbar.

Eine grundlegende leichtgewichtige Beschreibunglogik ist die Beschreibungslogik $\mathcal{EL}$. Anders als $\mathcal{ALC}$ sind in $\mathcal{EL}$ nur die Basisoperatoren $\sqcap$, $\exists r.C$ und $\top$ erlaubt. Diese verringerte Ausdrucksstärke hat allerdings den Vorteil, dass das Erfüllbarkeits- und Subsumtionsproblem in polynomieller Zeit entschieden werden können. So hat z.B.: (vgl. Brandt, 2004, S.299) gezeigt, dass in $\mathcal{EL}$ die Subsumtion mit generellen TBoxen in polynomieller Zeit entscheidbar ist.

Auch für verschiedene Erweiterungen von $\mathcal{EL}$ um weitere Operatoren gilt, dass diese noch effizient entscheidbar sind. Beispielhaft ist die Erweiterung $\mathcal{EL}^{++}$ zu nennen, die $\mathcal{EL}$ um $\bot$ (Bottom-Konzept), Konzeptassertion (ein Individuum ist eine Instanz eines Konzeptnamen), Rollenassertion und weitere Operatoren erweitert (Baader et al., 2005, vgl. S.3), sodass sie trotzdem noch effizient entscheidbar ist. $\mathcal{EL}^{++}$ ist als ein Profil in OWL 2 implementiert (Motik et al., 2012).

Für weitere Erweiterung von $\mathcal{EL}$ gilt, dass das Subsumtionsproblem nicht mehr effizient entscheidbar ist. Zu diesen Erweiterungen gehören z.B. $\mathcal{EL}^{(\neg)}$ (Erweiterung um primitive Negation) und $\mathcal{ELU}$ (Erweiterung um Disjunktion) (Baader et al., 2005, vgl. S.21). Eine dieser nicht-effizienten Erweiterungen ist die Logik $\mathcal{EL}^{\mathcal{F}}$, die $\mathcal{EL}$ um die Eigenschaft erweitert, dass alle Rollen funktional sind. Für diese Logik ist das Subsumtionsproblem mit azyklischen TBoxen co-NP-vollständig ist (vgl. Haase, 2007, S.57).

In der vorliegenden Arbeit werden wir das Subsumtionsproblem von $\mathcal{EL}^{\mathcal{F}}$ mit azyklischen TBoxen betrachten, und versuchen einen Reasoner für dieses Problem zu entwickeln und zu implementieren. Für die azyklischen TBoxen werden wir die Normalform nach Haase (2007, S.11) nutzen. Diese wird in Haase (2007, S.18) wie folgt beschrieben: Für jedes $A \equiv C \in \mathcal{T}$ muss C die Form $\top, B, \exists r.B_1$ oder $B_1 \sqcap B_2$ (mit $B \in N_C$ und $B_1, B_2 \in N_{def}(\mathcal{T})$) haben. Man sieht leicht, dass man das von Haase (2007, S.18) angegebne polynomielle Verfahren zum Übersetzen einer generellen $\mathcal{EL}$-TBox in diese Normalform auch für Terminologien mit funktionalen Rollen, wie $\mathcal{EL}^{\mathcal{F}}$-TBoxen, nutzen kann.

3 Theoretische Grundlagen

3.1 Charakterisierung von Subsumtion mittels Expansionsbäumen

Ab jetzt nutzen wir o.b.d.a. Konzeptnamen wenn wir über die bezeichneten Konzepte sprechen. In diesem Kapitel wollen wir zeigen, dass es möglich ist, zu zwei gegebenen Konzepten A, B und einer gegebenen azyklischen TBox $\mathcal{T}$ einen Baum zu bilden, sodass gilt: A wird von B subsumiert, genau dann wenn die Beschriftung aller Blätter des Baumes von B in den Beschriftungen aller Blätter des Baumes von A vorkommen. Die Idee und der dafür notwendige Beweis lehnt sich dabei an den Beweis von Nebel (1990, vgl. Kap.3 S. 240-241) an, der aber eine andersartige Logik ohne Funktionalität nutzt. Zuerst definieren wir $U_{\mathcal{T}}(A)$ als eine *Expansion* von A entlang einer gegebenen TBox $\mathcal{T}$, in Form eines Baumes. $U_{\mathcal{T}}(A)$ beschreibt dabei intuitiv die von einem Element $d \in A$ zu erfüllenden Axiome. Dabei wird davon ausgegangen, dass die gegebene TBox in der Normalform aus dem Kapitel 2.2 vorliegt. Danach definieren wir $leaves(U_{\mathcal{T}}(A))$ intuitiv, als die Blätter des definierten Baumes.

Sei $U_{\mathcal{T}}(A) = (V_A, E_A, L, K)$ ein gerichteter und beschrifteter Graph mit L eine Menge von Kantenbeschriftungen: $L = \{\exists, \sqcap\}$ und K eine Menge von Knotenbeschriftungen. V_A ist dabei die Menge der Knoten, $E_A \subseteq V_A \times L \times V_A$.

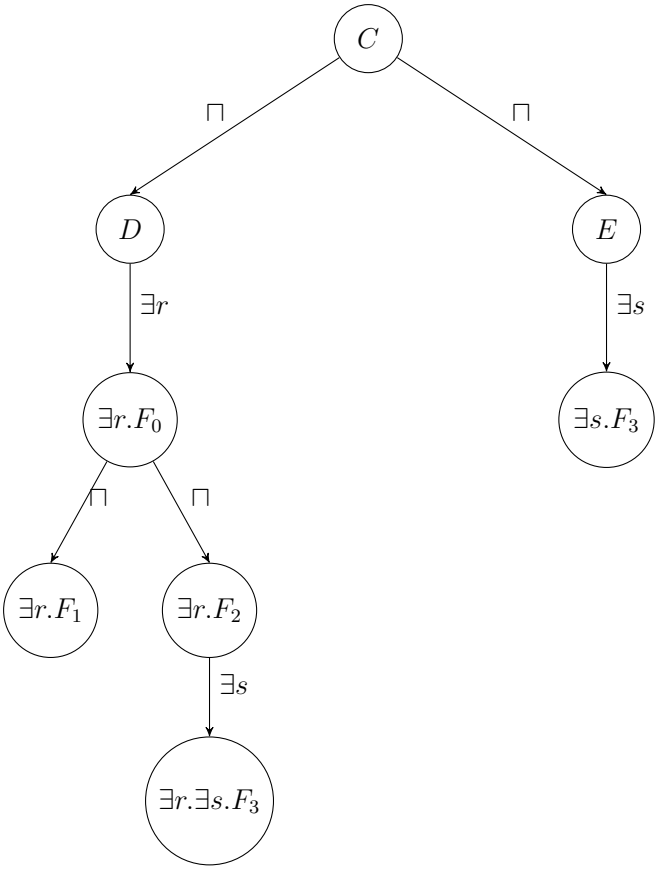
Nun definiere $U_{\mathcal{T}}(A)$ iterativ wie folgt: Zu Beginn besteht der Baum $U_{\mathcal{T}}(A)$ nur aus ein Knoten, beschriftet mit A: $U_{\mathcal{T}}(A) = (\{a\}, \emptyset, \emptyset, \{a \to A\})$. Für jedes Blatt b im aktuellen Baum $U_T(A)$ dessen Konzeptsuffix der Knotenbeschriftung C kein primitives Konzept beschreibt:

1. Betrachte das Konzeptsuffix B von der Beschriftung $\exists r_1 \ldots \exists r_k.B$ (mit $k \geq 0$) vom Blatt b. Für B gilt:
   - 1.1. Wenn für B gilt $B \equiv B_1 \sqcap B_2 \in \mathcal{T}$: Füge b_1, b_2 zu V, $\{b_1 \to \exists r_1 \ldots \exists r_k.B_1, b_2 \to \exists r_1 \ldots \exists r_k.B_2\}$ zu K und $\{(a, \sqcap, b_1), (a, \sqcap, b_2)\}$ zu E von $U_{\mathcal{T}}(A)$ hinzu.
   - 1.2. Wenn für B gilt, $B \equiv \exists r.B_1 \in \mathcal{T}$: Füge b_1 zu V und $(b, \exists r, b_1)$ zu E von $U_{\mathcal{T}}(A)$ hinzu. Außerdem füge die Beschriftung $\{b1 \to \exists r_1 \ldots \exists r_k.\exists r.B_1\}$ zu K hinzu.
   - 1.3. Wiederhole dies solange, bis für alle Blätter von $U_{\mathcal{T}}(A)$ gilt, dass sie primitive Konzepte als Knotensuffix bei den Knotenbeschriftung besitzen.

Als Beispiel betrachten wir $U_{\mathcal{T}}(C)$ für die folgenden TBox $\mathcal{T}$:

$$\begin{aligned} C &\equiv D \sqcap E \\ D &\equiv \exists r.F_0 \\ F_0 &\equiv F_1 \sqcap F_2 \\ F_2 &\equiv \exists s.F_3 \\ E &\equiv \exists s.F_3 \\ G &\equiv F_1 \sqcap F_2 \\ H &\equiv \exists r.G \end{aligned} \tag{2}$$

$U_{\mathcal{T}}(C)$ würde nach dem iterativen Aufbau nun folgenden Baum darstellen:



Nun definieren wir $leaves(U_{\mathcal{T}}(A))$ als Beschriftungen aller Blätter von $U_{\mathcal{T}}(A)$.

Im Folgenden bezeichnen wir Konzepte der Form $\exists r_0. \ldots \exists r_n A$, mit $r_0, \ldots, r_n \in N_R$ und $A \in N_C$ ein primitivs Konzepts, als lineare Konzepte. Intuitiv entsprechen die Blätter der Bäume den linearen Konzepten der Formeln. Am Beispiel C gilt also $leaves(U_{\mathcal{T}}(C)) = \{\exists r.F_1, \exists s.F_3, \exists s.\exists r.F_3\}$. Im Folgenden soll folgendes Theorem gezeigt werden:

Theorem 1. *Für alle azyklischen TBoxen $\mathcal{T}$ und Konzeptnamen A_0, B_0 gilt:*

$A_0 \sqsubseteq_{\mathcal{T}} B_0 \Leftrightarrow leaves(U_{\mathcal{T}}(B_0)) \subseteq leaves(U_{\mathcal{T}}(A_0))$

Um Theorem 1 zu zeigen, wird zunächst folgendes Lemma gezeigt:

Lemma 2. *Für alle Konzeptnamen A, für alle $\mathcal{I} \models \mathcal{T}$ gilt:*

$A^{\mathcal{I}} = (\bigsqcap leaves(U_{\mathcal{T}}(A)))^{\mathcal{I}}$

Beweis. Das Lemma 2 zeigen wir induktiv über Position von A in $\mathcal{T}$:

IA.: A primitiv in $\mathcal{T}$. Dann gilt

$$(\bigsqcap leaves(U_{\mathcal{T}}(A)))^{\mathcal{I}} \overset{Konstr.}{=} (\bigsqcap \{A\})^{\mathcal{I}} \overset{Sem.}{=} A^{\mathcal{I}}$$

Die erste Gleichung gilt nach der Konstruktion von $U_{\mathcal{T}}(A)$. Die zweite Gleichung gilt aufgrund der Semantik von $\bigsqcap$ über eine einelementige Menge.

IS.: Fallunterscheidung

1. Fall: $A \equiv B_1 \sqcap B_2 \in \mathcal{T}$. Dann gilt:

$$\begin{aligned}(\bigsqcap leaves(U_\mathcal{T}(A))^\mathcal{I} &\overset{Konstr.}{=} (\bigsqcap(leaves(U_\mathcal{T}(B_1)) \cup leaves(U_\mathcal{T}(B_2)))^\mathcal{I} \\ &\overset{Sem.}{=} ((\bigsqcap(leaves(U_\mathcal{T}(B_1)))) \sqcap (\bigsqcap leaves(U_\mathcal{T}(B_2))))^\mathcal{I} \\ &\overset{IV.}{=} (B_1 \sqcap B_2)^\mathcal{I} \\ &\overset{Def.}{=} A^\mathcal{I}\end{aligned} \tag{3}$$

   Die erste Gleichung gilt nach der Konstruktion von $U_\mathcal{T}(A)$. Die zweite Gleichung gilt aufgrund der Semantik von $\bigsqcap$, die dritte Gleichung aufgrund der Induktionsvoraussetzung und die letze Gleichung aufgrund der Definition $A \equiv B_1 \sqcap B_2 \in \mathcal{T}$.

2. Fall: $A \equiv \exists r.B \in \mathcal{T}$. Dann gilt:

$$\begin{aligned}(\bigsqcap leaves(U_\mathcal{T}(A))^\mathcal{I} &\overset{Konstr.}{=} (\bigsqcap\{\exists r.C | C \in leaves(U_\mathcal{T}(B))\})^\mathcal{I} \\ &\overset{Funk.}{=} (\exists r.(\bigsqcap leaves(U_\mathcal{T}(B)))^\mathcal{I} \\ &\overset{IV.}{=} (\exists r.B)^\mathcal{I} \\ &\overset{Def.}{=} A^\mathcal{I}\end{aligned} \tag{4}$$

   Die erste Gleichung gilt nach der Konstruktion von $U_\mathcal{T}(A)$. Die zweite Gleichung gilt, da aufgrund der Funktionalität von Rollen $(\exists s.B_1 \sqcap \ldots \sqcap \exists s.B_n))^\mathcal{I} = (\exists s.(B_1 \sqcap \ldots \sqcap B_n))$ gelten muss. Danach kann wie im vorherigen Fall über die Induktionsvoraussetzung und der Definition argumentiert werden.

□

Nun betrachte Theorem 1:

$$A_0 \sqsubseteq_\mathcal{T} B_0 \Leftrightarrow leaves(U_\mathcal{T}(B_0)) \subseteq leaves(U_\mathcal{T}(A_0))$$

Beweis. $\Leftarrow$

Sei $\mathcal{I} \models \mathcal{T}$. Dann gilt die Behauptung, da:

$$\begin{aligned}A_0^\mathcal{I} &\overset{Lem.2}{=} (\bigsqcap leaves(U_\mathcal{T}(A_0)))^\mathcal{I} \\ &\overset{Vor.}{\subseteq} (\bigsqcap leaves(U_\mathcal{T}(B_0)))^\mathcal{I} \\ &\overset{Lem.2}{=} (B_0)^\mathcal{I}\end{aligned} \tag{5}$$

Die erste und letzte Gleichung gilt aufgrund des Lemma 2. Die Teilmengenbeziehung gilt aufgrund der Vorraussetzung von dieser Richtung des Beweises.

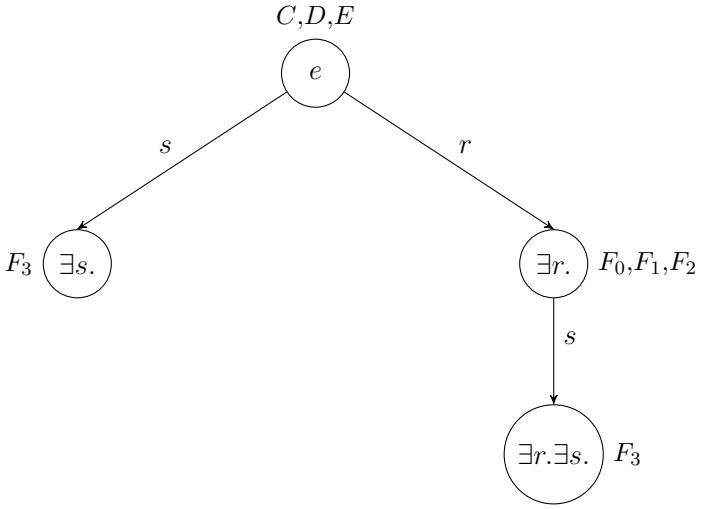
$\Rightarrow$

Diese Richtung zeigen wir per Kontraposition: Also angenommen $leaves(U_\mathcal{T}(B_0)) \not\subseteq leaves(U_\mathcal{T}(A_0))$, dann gibt es ein lineares Konzept $\exists r_1 \cdots \exists r_n.B \in leaves(U_\mathcal{T}(B_0)) \setminus leaves(U_\mathcal{T}(A_0))$ mit $n \geq 0$.

Konstr. $\mathcal{J}$ aus $leaves(U_\mathcal{T}(A_0))$, sodass $\mathcal{J} \models \mathcal{T}$ und $\mathcal{J} \not\models_\mathcal{T} A_0 \sqsubseteq B_0$ gilt. Dafür konstruieren wir zunächst die Interpretation $\mathcal{I}$, die intuitiv gesprochen alle Axiome "vorwärts gelesen" erfüllen soll (also für ein Axiom der Form $A \equiv B$ wollen wir das $A \sqsubseteq B$ erfüllt wird) und eine Abbildung $f : V_{U_\mathcal{T}(A_0)} \to \Delta^\mathcal{I}$ von den Knoten von $U_\mathcal{T}(A_0)$) zu den Elementen der Domäne, mit den folgenden Schritten:

1. Füge ein Element e zur Domäne $\Delta^\mathcal{I}$ und zu $A_0^\mathcal{I}$ hinzu.
2. Für jedes lineare Konzept in $leaves(U_\mathcal{T}(A_0))$ der Form $\exists r_1. \ldots \exists r_i.$ füge jedes Teilpräfix der Form $\exists r_1. \ldots \exists r_k.$ (mit $2 \leq k \leq i$) als ein Element mit dem Namen $r_1. \ldots r_k.$ zur Domäne $\Delta^\mathcal{I}$ hinzu, falls für diese Rollenkette noch kein Element in der Domäne hinzugefügt wurde. Zudem füge für jeden Knoten $v \in V_{U_\mathcal{T}(A_0)}$ von $U_\mathcal{T}(A_0)$ dessen Beschriftung die Form $\exists r_1. \ldots \exists r_k.E$, mit E ein beliebiger Konzeptname in N_C, hat, die Abbildung $(v \to r_1. \ldots r_k.)$ zu f hinzu.
3. Nun gehe für jedes hinzugefügte Element mit einem Namen der Form $r_1, \ldots, r_k$ wie folgt vor:
   3.1. Definiere e als Element für das leere Rollenpräfix. Für jeden Knoten $v_0 \in V_{U_\mathcal{T}(A_0)}$ von $U_\mathcal{T}(A_0)$ für den es eine Abbildung $f(v_0) = d_1$ und ein Knoten $v_1 \in V_{U_\mathcal{T}(A_0)}$ gibt, sodass es eine Kante zwischen v_0 und v_1 in $U_\mathcal{T}(A_0)$ mit der Beschriftung $\exists.r$ (mit $r \in N_R$) gibt, füge $(d_1, f(v_1))$ zu $(r_k)^\mathcal{I}$ hinzu.
4. Nun gehe für die Elemente der Domäne, beginnend mit e, wie folgt vor:
   4.1. Gegeben das Element d_0. Für jeden Konzeptnamen D mit $d_0 \in D^\mathcal{I}$ betrachte alle Axiome in $\mathcal{T}$ mit D auf der linken Seite (da $\mathcal{T}$ azyklisch gibt es ≤ 1 Axiom dieser Form):
      4.1.1. Wenn $D \equiv E \sqcap F$, dann füge d_0 zu $E^\mathcal{I}$ und $F^\mathcal{I}$ hinzu.
      4.1.2. Wenn $D \equiv \exists r.E$ mit $r \in \{r_1. \ldots r_i.\}$ und es ein d_1 mit $(d_0, d_1) \in r^\mathcal{I}$ gibt (das nach Konstruktion eindeutig bestimmt ist), dann füge das Element d_1 zu $E^\mathcal{I}$ hinzu, und beginne den Schritt 4.1.1 mit d_1

Die so gebildete Interpretation $\mathcal{I}$ für unsere Beispiel-TBox würde daher wie folgt für das Konzept C aussehen:



mit $leaves(U_\mathcal{T}(C)) = \{\exists r.F_1, \exists s.F_3, \exists s.\exists r.F_3\}$

Dieser Algorithmus terminiert, denn: Schritt 1 terminiert offensichtlich. Schritt 2 und 3 iterieren über eine Menge von Elementen und terminieren daher auch. Der 4. Schritt terminiert, da wir nur endlich viele Elemente in der Domäne und endlich viele Konzeptnamen haben, und maximal für jedes Element jeden Konzeptnamen betrachten.

Die Abbildung f ist nach Konstruktion offenbar für alle Knoten aus $U_{\mathcal{T}}(A_0)$ definiert, da nach Konstruktion von $U_{\mathcal{T}}(A_0)$ jeder Knoten ein Rollenpräfix besitzen muss, dass als Teilpräfix in den Rollenpräfixen der linearen Konzepten vorkommt.

Nun wollen wir für diese gebildete Interpretation $\mathcal{I}$ folgende Punkte zeigen:

(a) $e \in A_0^{\mathcal{I}}$

(b) $\mathcal{I} \models$ "$\mathcal{T}$ vorwärts" (also $\mathcal{I} \models A \sqsubseteq B$ für jedes Axiom $A \equiv B \in \mathcal{T}$)

(c) $e \notin B_0^{\mathcal{I}}$

(d) Alle $r^{\mathcal{I}} \in N_R$ sind funktional

Punkt a gilt offensichtlich nach Konstruktion (Schritt 1). Nun zeigen wir noch Punkt b und c:

(b) *Beweis.* Um dies zu zeigen, müssen wir für jedes $A \equiv C$ mit $A \in N_C$ und C ein zusammengesetztes Konzept zeigen, dass $\mathcal{I} \models A \sqsubseteq C$ von $\mathcal{I}$. Betrachten wir ein beliebiges Element $d_1 \in \Delta^{\mathcal{I}}$ für das gilt $d_1 \in A^{\mathcal{I}}$.

Wenn $C = D \sqcap E$ mit $D, E \in N_C$, dann hätten wir d_1 im Schritt 4.1.1 auch zu $D^{\mathcal{I}}$ und $E^{\mathcal{I}}$ hinzugefügt, also wäre $A \sqsubseteq C$ an d_1 erfüllt.

Wenn $C = \exists r.D$ mit $r \in N_R$ und $D \in N_C$, dann existiert ein eindeutiges d_2 mit $(d_1, d_2) \in r^{\mathcal{I}}$, da C in $U_{\mathcal{T}}(A_0)$ als Konzeptsuffix für einen Knoten $v_0 \in V_{U_{\mathcal{T}}(A_0)}$ vorkommt und nach Konstruktion im Schritt 1.2 von $U_{\mathcal{T}}(A_0)$ eine Kante von v zu einem weiteren Knoten $v_1 \in V_{U_{\mathcal{T}}(A_0)}$ existiert mit der Beschriftung $\exists r.$. Aufgrund dieser Kante gibt es nach Schritt 3.1 ein Element $(d_1, d_2) \in r^{\mathcal{I}}$. Nach Schritt 4.1.2 musste d_2 nun zu $E^{\mathcal{I}}$ hinzugefügt worden sein. Also erfüllt d_1 $C = \exists r.D$. □

(c) *Beweis.* Angenommen es gilt $e \in B_0$. Dann muss e im Schritt 4 zu B_0 hinzugefügt worden sein. Da e in unserer Konstruktion das erste Element ist, und für jede Rolle r aufgrund der Azyklität der TBox gilt, dass keine "Rollenkante" der Form (d_i, e) in $r^{\mathcal{I}}$ vorkommt, muss e im Schritt 4.1.1 zu B_0 hinzugefügt wurde. Dies kann aber nur durch ein Axiom der Form $A_0 \equiv B_0 \sqcap C$ mit C ein Konzeptname geschehen sein, da e nach Konstruktion im Schritt 1 zu A_0 hinzugefügt wurde. Dann müsste aber $leaves(U_{\mathcal{T}}(B_0)) \subseteq leaves(U_{\mathcal{T}}(A_0))$, da $U_{\mathcal{T}}(B_0)$ nach Konstruktion ein Teilbaum von $U_{\mathcal{T}}(A_0)$ sein müsste. □

(d) *Beweis.* Während der Konstruktion fügen wir für jedes $e \in \Delta^{\mathcal{I}}$ und für jedes $r^{\mathcal{I}} \in N_R$ ein (d, e) zu $r^{\mathcal{I}}$ hinzu. □

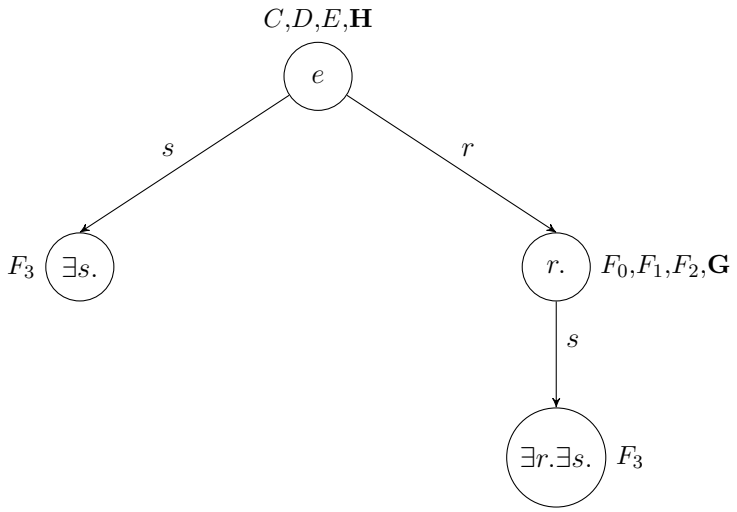
Nun haben wir gezeigt, dass alle Axiome der Form $A \equiv B \in \mathcal{T}$ '"vorwärts" erfüllt wurden, ein Model muss diese Axiome aber auch '"rückwärts" (also $B \sqsubseteq A$) erfüllen. Daher definieren wir $\mathcal{J}$ als Erweiterung von $\mathcal{I}$ (zu Beginn setzen wir also $\mathcal{J} = \mathcal{I}$ und erweitern dies dann). Zur Erweiterung werden alle Axiome aus der TBox $\mathcal{T}$ '"rückwärts" erschöpfend angewendet und die Interpretationen der Konzeptnamen um die entsprechenden Elemente erweitert.

5. Für jedes Axiom $A \equiv C \in \mathcal{T}$:

5.1. Füge jedes Element d_0 aus der Domäne $\Delta^{\mathcal{J}}$ für das C erfüllt ist, und für das gilt $C \notin A^{\mathcal{J}}$, zu $A^{\mathcal{J}}$ hinzu.

5.2. Wiederhole dies, bis sich nach einen Durchlauf durch alle Axiome für jedes Konzeptname $A \in N_C$ die Menge $A^{\mathcal{J}}$ sich nicht geändert hat.

Die so erweiterte Interpretation $\mathcal{J}$ von $\mathcal{I}$ unserer Beispiel-TBox würde daher wie folgt aussehen (Änderungen zu $\mathcal{I}$ sind fett-gedruckt):



mit $leaves(U_{\mathcal{T}}(C)) = \{\exists r.F_1, \exists s.F_3, \exists s.\exists r.F_3\}$s

Auch dieser Algorithmus terminiert, denn wir haben nur endlich viele Elemente, und endlich viele Konzeptnamen, und würden so spätestens bei den Punkt, bei dem für alle Konzeptnamen A gilt $A^{\mathcal{J}} = \Delta^{\mathcal{I}}$ terminieren.

Für $\mathcal{J}$ wollen wir nun folgende Punkte zeigen.

1. $d_0 \in A_0^{\mathcal{J}}$
2. $\mathcal{J} \models \mathcal{T}$
3. $d_0 \notin B_0^{\mathcal{J}}$
4. Alle $r^{\mathcal{J}} \in N_R$ sind funktional

Punkt (a) gilt wieder offensichtlich, da $\mathcal{J}$ eine Erweiterung von $\mathcal{I}$ ist, also gilt $A_0^{\mathcal{I}} \subseteq A_0^{\mathcal{J}}$ mit $d_0 \in A_0^{\mathcal{I}}$ also auch $d_0 \in A_0^{\mathcal{J}}$. Punkt (d) gilt, da wir die mit den Rollennamen beschriebenen Mengen im 5. Schritt nicht ändern. Nun müssen wir noch Punkt (b) und Punkt (c) zeigen

(b) *Beweis.* Für jedes Axiom $A \equiv C \in \mathcal{T}$ mit $A \in N_C$ und C ist die Richtung $A \sqsubseteq C$ von $\mathcal{J}$ erfüllt: Für jedes Element $d \in A^{\mathcal{J}}$ gilt:

- Wenn $d \in A^{\mathcal{I}}$, dann gilt auch $d \in C^{\mathcal{J}}$, da $\mathcal{J}$ eine Erweiterung von $\mathcal{I}$ ist und wir $\mathcal{I} \models$ "$\mathcal{T} vorwaerts$" gezeigt haben.
- Wenn $d \notin A^{\mathcal{I}}$, dann wurde d zu $A^{\mathcal{J}}$ durch Schritt 5.1 hinzugefügt. Da T aber eine azyklische TBox ist, kommt A nur in max. einem Axiom von $\mathcal{T}$ auf der linken Seite vor. Dieses muss dann $A \equiv C$ sein. Daher muss $d \in C^{\mathcal{J}}$ bereits vor der Anwendung der Regel gegolten haben.

Nach dem Schritt 5.1 gilt aber auch das $C \sqsubseteq A$ von $\mathcal{J}$ erfüllt wird, da wir jedes Element der Domäne das C erfüllt zu $A^{\mathcal{J}}$ hinzufügen (und $A \in N_C$ da T in Normalform). Da in der azyklischen TBox $\mathcal{T}$ nur ein Axiom mit A auf der linken Seite vorkommt, muss das für Schritt 5.1 genutzte Axiom eben dieses sein, sodass das Element bereits vor der Anwendung in $C^{\mathcal{J}}$ gewesen sein muss.

Da für jedes $A \equiv C \in \mathcal{T}$ die Axiome $A \sqsubseteq C$ und $C \sqsubseteq A$ von $\mathcal{J}$ erfüllt werden, wird auch jedes $A \equiv C \in \mathcal{T}$ von $\mathcal{J}$ erfüllt. Daher gilt $\mathcal{J} \models \mathcal{T}$. □

(c) *Beweis.* Wir zeigen $e \notin (B_0)^{\mathcal{J}}$ per Widerspruch:

- Angenommen $e \in (B_0)^{\mathcal{J}}$, dann muss nach Lemma 2 gelten: $e \in (\bigsqcap leaves(U_{\mathcal{T}}(B_0)))^{\mathcal{I}}$.
- Dann gilt $e \in (\exists r_1. \ldots \exists r_n.B)^{\mathcal{I}}$ mit $n \geq 0$ für jedes lineare Konzept in $leaves(U_{\mathcal{T}}(B_0)))^{\mathcal{I}}$.
- Dann gilt $r_1. \ldots r_n \in B^{\mathcal{J}}$ nach der Konstruktion von $\mathcal{I}$ im Schritt 4.
- Da B primitiv ist, muss auch $r_1. \ldots r_n. \in B^{\mathcal{I}}$ gelten (da es kein Axiom der Form $B \equiv E \in \mathcal{T}$ geben kann).
- Dann muss $r_1. \ldots r_n.$ durch Schritt 4 der Konstruktion zu $B^{\mathcal{I}}$ hinzugefügt worden. Daher muss eine Rollenkette zu $r_1. \ldots r_n.$ führen.
- Dann muss in $U_{\mathcal{T}}(B_0)$ Knoten mit den Rollenpräfix $\exists r_1. \ldots \exists r_n.$ existieren (siehe Schritt 3.1). Einer von diesen muss die Beschriftung $\exists r_1. \ldots \exists r_n.B$ haben, da wir sonst beim Aufbau über 4 nicht dazu gekommen wären das entsprechende Element zu $B^{\mathcal{I}}$ hinzuzufügen.
- Dann gilt aber $leaves(U_{\mathcal{T}}(B_0)) \subseteq leaves(U_{\mathcal{T}}(A_0))$ gelten, was im Widerspruch zur Vorraussetzung steht.

□

Da $\mathcal{J} \models \mathcal{T}$ und $\mathcal{J} \not\models_{\mathcal{T}} A_0 \sqsubseteq B_0$, folgt das $A \not\sqsubseteq_{\mathcal{T}} B_0$. □

3.2 Reduktion auf SAT

In diesem Kapitel bilden wir die aussagenlogische Formel $\varphi^{\mathcal{T}}_{A_0,B_0}$, die erfüllbar sein soll, wenn die Blätter von $U_{\mathcal{T}}(B_0)$ nicht in den Blättern von $U_{\mathcal{T}}(A_0)$ enthalten sind. Aufgrund des Theorem 1 gilt dann zudem $A_0 \not\sqsubseteq B_0$.

Dazu werden zunächst Formeln definiert, die die korrekte Struktur des Baumes festhalten soll. Zudem werden Formeln definiert, die einen Pfad in $U_{\mathcal{T}}(B_0)$ von der Wurzel zu einem Blatt darstellen soll. Zum Schluß wird eine Formel festgehalten, mit der ausgedrückt werden sollen, dass $U_{\mathcal{T}}(B_0)$ ein lineares Konzept als Blatt beinhaltet, dass nicht in $U_{\mathcal{T}}(A_0)$ als Blatt vorkommt.

3.2.1 Bilden der aussagenlogischen Formeln

Für dieses Kapitel benötigen wir zunächst die Formel $depth_{\mathcal{T}}(B_0)$ (kurz: $d^{B_0}_{\mathcal{T}}$), die die maximale Höhe des Baumes $U_{\mathcal{T}}(B_0)$, und $roledepth_{\mathcal{T}}(B_0)$ (kurz: $r^{B_0}_{\mathcal{T}}$), die die maximale Rollentiefe von $\mathcal{T}$ beschreibt.

Zunächst definieren wir $depth_{\mathcal{T}}(B_0)$ induktiv wie folgt:

- B_0 primitiv in $\mathcal{T}$: $depth_{\mathcal{T}}(B_0) = 0$
- $B_0 \equiv C_0 \sqcap C_1 \in \mathcal{T}$: $depth_{\mathcal{T}}(B_0) = 1 + \max(depth_{\mathcal{T}}(C_0), depth_{\mathcal{T}}(C_1))$
- $B_0 \equiv \exists r.D_0 \in \mathcal{T}$: $depth_{\mathcal{T}}(B_0) = 1 + depth_{\mathcal{T}}(D_0)$

$roledepth_{\mathcal{T}}(B_0)$ wird definiert wie folgt:

- B_0 primitiv in $\mathcal{T}$: $roledepth_{\mathcal{T}}(B_0) = 0$
- $B_0 \equiv D_0 \sqcap D_1 \in \mathcal{T}$: $roledepth_{\mathcal{T}}(B_0) = \max(roledepth_{\mathcal{T}}(D_0), roledepth_{\mathcal{T}}(D_1))$
- $B_0 \equiv \exists r.E_0 \in \mathcal{T}$: $roledepth_{\mathcal{T}}(B_0) = 1 + roledepth_{\mathcal{T}}(E_0)$

Um das gegebene Problem auf SAT zu reduzieren, werden wir zunächst verschiedene Aussagenlogische Formeln definieren. Dazu nutzen wir aussagenlogische Variabeln der Form $p_{\ell,i,S}$ und $q_{\ell,i}$. Die $p_{\ell,i,S}$ für die $V(p_{\ell,i,S}) = \text{true}$ gilt, beschreiben einen Pfad in $U_{\mathcal{T}}(B_0)$ von der Wurzel zu einem linearen Konzept. Dabei sagt $p_{\ell,i,S}$, dass in dem beschriebenen Pfad in der ℓ. Ebene des Baumes and der i. Stelle S steht, mit $S \in N_C \cup N_R$. Beispielsweise würde, wenn $V(p_{3,1,r}) = \text{true}$ und $V(p_{3,1,F_0}) = \text{true}$, der Knoten mit der Beschriftung $\exists r.F_0$ in den Pfad von C nach $\exists r.F_1$ (oder $\exists s.\exists r.F_3$) beschrieben werden.

Nun definieren wir einzelne Teilformeln für $\varphi^{\mathcal{T}}_{A_0,B_0}$. Beispielsweise definieren wir die Formel ψ_1 um festzuhalten, dass auf jedem Ebene und jeder Stelle des Pfades nur ein $S \in N_C \cup N_R$ vorkommt, um die korrekte Form des Pfades (und des Baumes) festzuhalten. Dazu sagt die Formel aus, dass wenn $V(p_{\ell,i,S_1}) = \text{true}$ (mit $S_1 \in N_C \cup N_R$) alle anderen p_{ℓ,i,S_2} mit $S_2 \in N_C \cup N_R$ und $S_1 \neq S_2$ falsch sein müssen.

Als ein anderes Beispiel betrachten wir die Formel ψ_4. Diese soll aussagen, dass wenn auf einen Ebene an einer Stelle ein Konzeptname $A \in N_C$ steht, an den nächsten Stelle kein weiteres $S \in N_C \cup N_R$ vorkommen kann, da Beschriftungen immer die Form $\exists r_1 \ldots r_n.A$ mit $r_0, \ldots, r_n \in N_R$ und $A \in N_C$ hat.

Zunächst beschreiben wir die Formel $repres^{\mathcal{T}}_{B_0}$. Diese setzt sich aus verschiedenen Konjunkten zusammen, die wir einzeln betrachten:

Zuerst betrachten wir die Konjunkte, die Sicherstellen sollen, dass für jedes ℓ und jedes i die $p_{\ell,i,X}$ die Pfade in $U_{\mathcal{T}}(B_0)$ korrekt repräsentieren:

1. Es gibt genau ein A oder $\exists.r$ auf jeder gegebenen Position:

$$\psi_1 = \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{S \in N_C \cup N_R} \left(p_{\ell,i,S} \rightarrow \bigwedge_{B \neq S \in N_C \cup N_R} \neg p_{\ell,i,B} \right)$$

2. Auf jedem $\exists.r$ folgt entweder ein $\exists.r'$ oder A:

$$\psi_2 = \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{r \in N_R} \left(p_{\ell,i,r} \rightarrow \bigvee_{A \in N_C} p_{\ell,i+1,A} \vee \bigvee_{r' \in N_R} p_{\ell,i+1,r'} \right)$$

3. Immer wenn auf einer Ebene ℓ and Stelle i eine Rolle r vorkommt, muss im nächsten Ebene $\ell + 1$ and der Stelle i das gleiche r stehen, oder nichts (also kein $p_{\ell,i,X}$ darf erfüllt sein):

$$\psi_3 = \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{r \in N_R} \left(p_{\ell,i,r} \rightarrow (p_{\ell+1,i,r} \vee \bigwedge_{S \in N_C \cup N_R} \neg p_{\ell+1,i,s}) \right)$$

4. Auf jeden Konzeptnamen A folgt kein weiteres Element:

$$\psi_4 = \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{A \in N_C} \left(p_{\ell,i,A} \rightarrow (\bigwedge_{A' \in N_C} \neg p_{\ell,i+1,A'} \wedge \bigwedge_{r'} \neg p_{\ell,i+1,r'}) \right)$$

5. Wenn auf einem Ebene ℓ an einer Stelle i kein p wahr ist, muss dies auch an der folgenden Stelle $i + 1$ für alle p gelten ("Auf jedem nichts folgt nichts"):

$$\psi_5 = \bigwedge_{l \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \left((\bigwedge_{A \in N_C} \neg p_{l,i,A} \wedge \bigwedge_{r \in N_R} \neg p_{\ell,i,r}) \rightarrow (\bigwedge_{A' \in N_C} \neg p_{\ell,i+1,A'} \wedge \bigwedge_{r' \in N_R} \neg p_{\ell,i+1,r'}) \right)$$

6. Wenn auf einem Ebene ℓ an der 0. Stelle kein p wahr ist, muss dies auch an der 0. Stelle der nächsten Ebene für alle p gelten. Dies ist im Grunde eine Variation von ψ_5 ("Auf jedem nichts folgt nichts"):

$$\psi_6 = \bigwedge_{l \leq d_{\mathcal{T}}(B_0)} \left((\bigwedge_{A \in N_C} \neg p_{\ell,0,A} \wedge \bigwedge_{r \in N_R} \neg p_{\ell,0,r}) \rightarrow (\bigwedge_{A' \in N_C} \neg p_{\ell+1,0,A'} \wedge \bigwedge_{r' \in N_R} \neg p_{\ell+1,0,r'}) \right)$$

7. Wenn ein Blatt im Baum erreicht wurde, dann kann es in der nächsten Ebene keinen weiteren Knoten geben, also kein p erfüllt werden (es reicht dies nur für die 0. Stelle des Ebenes zu sagen, da ψ_5 dafür sorgt, dass darauf auch kein p erfüllt sein kann):

$$\psi_7 = \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{A \in N_C \, mit \, A \, primitiv} \left(p_{\ell,i,A} \rightarrow (\bigwedge_{A' \in N_C} \neg p_{\ell+1,0,A'} \wedge \bigwedge_{r' \in N_R} \neg p_{\ell+1,0,r'}) \right)$$

Aus diesen Konjunkten setzt sich $repres^{\mathcal{T}}_{B_0}$ zusammen:

$$repres^{\mathcal{T}}_{B_0} = \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4 \wedge \psi_5 \wedge \psi_6 \wedge \psi_7$$

Nun wollen wir zudem $path^{\mathcal{T}}_{B_0}$ definieren, dass ein Pfad innerhalb von $U_{\mathcal{T}}(B_0)$ beschreiben soll. Zunächst wird in ψ_8 festgehalten, dass B_0 die Beschriftung der Wurzel ist. Dazu benötigen wir folgende Konjunkte:

8. Auf der 0. Ebene an Stelle 0 steht B_0:

$$\psi_8 = p_{0,0,B_0}$$

9. Wenn auf Ebene ℓ, an beliebiger Stelle ein Konzeptname C vorkomment, der als $C \equiv C_1 \sqcap C_2$ in der TBox $\mathcal{T}$ vorkommt, dann kommt im nächsten Ebene an der Stelle der Konzeptname C_1 oder C_2 vor:

$$\psi_9 = \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{C \equiv C_1 \sqcap C_2 \in \mathcal{T}} (p_{\ell,i,C} \rightarrow p_{\ell+1,i,C_1} \vee p_{\ell+1,i,C_2})$$

10. Wenn auf Ebene ℓ, an beliebiger Stelle ein Konzeptname C vorkommnt, der als $C \equiv \exists r.C'$ in der TBox $\mathcal{T}$ vorkommt, dann kommt im nächsten Ebene an der gegebenen Stelle i r vor und an Stelle i + 1 kommt C' vor:

$$\psi_{10} = \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{C \equiv \exists r.C' \in \mathcal{T}} (p_{\ell,i,C} \rightarrow p_{\ell+1,i,r} \wedge p_{\ell+1,i+1,C'})$$

Wir setzen $path^{\mathcal{T}}_{B_0}$ wie folgt:

$$path^{\mathcal{T}}_{B_0} = \psi_8 \wedge \psi_9 \wedge \psi_{10}$$

Zudem benötigen wir die Formel $missing^{\mathcal{T}}_{A_0,B_0}$. Diese benutzt $q_{i,A}$, die aussagt, dass gegeben ein fixes $\exists$-Präfix $\exists r_1 \ldots \exists . r_i .$, das lineare Konzept $\exists r_1 \ldots r_i . A$ in $U_{\mathcal{T}}(A_0)$ vorkommt. $missing^{\mathcal{T}}_{A_0,B_0}$ ist definiert als:

$$missing^{\mathcal{T}}_{A_0,B_0} = \psi_{11}(k,i,A) \wedge \psi_{12}(k,i,A) \wedge \bigvee_{k \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigvee_{A \in N_C \, mit \, A \, primitiv} (\psi_{13}(k,i,A) \psi_{14} \wedge \psi_{15}(k,i,A))$$

$$\bigwedge_{\ell \leq d_{\mathcal{T}}(A_0), i \leq r_{\mathcal{T}}(A_0)} \bigwedge_{B \equiv \exists r.B' \in \mathcal{T}} (p_{\ell,i,r} \wedge q_{i,B} \rightarrow q_{i+1,B'})$$

mit $\psi_{10}, \ldots, \psi_{14}$ wie folgt:

11. für alle Konzepte B die als $B \equiv B_1 \sqcap B_2$ in der TBox $\mathcal{T}$ vorkommen und alle j die kleiner oder gleich i sind, gilt dass wenn $\exists r_1 \ldots \exists r_m . B$ in der j. Ebene als Knotenbeschriftung vorkommt, so auch $\exists r_1 \ldots \exists r_m . B_1$ und $\exists r_1 \ldots \exists r_m . B_2$:

$$\psi_{11}(k,i,A) = \bigwedge_{B \equiv B_1 \sqcap B_2 \in \mathcal{T}} \bigwedge_{j \leq i} (q_{j,B} \rightarrow q_{j,B_1} \wedge q_{j,B_2})$$

12. für alle Konzepte B die als $B \equiv \exists r.B'$ in der TBox $\mathcal{T}$ vorkommen und alle j die kleiner oder gleich i sind, gilt, dass wenn im in repres dargestellten Baum $U_{\mathcal{T}}(B_0)$ im Ebene k an der i. Stelle ein $\exists$.r vorkommt und , dass $\exists r_1 \ldots \exists r_j . B$ in der j. Ebene als Knotenbeschriftung vorkommt, dann kommt B' an der (j + 1)-ten Stelle als $\exists$-präfix vor ($\exists r_1 \ldots \exists r_{j+1} . B$):

$$\psi_{12}(k,i,A) = \bigwedge_{B \equiv \exists r.B' \in \mathcal{T}} \bigwedge_{j \leq i} (p_{k,j,r} \wedge q_{j,B} \rightarrow q_{j+1,B'})$$

13. im in repres dargestellten Baum $U_{\mathcal{T}}(B_0)$, gilt in einem Ebene k, dass an der i. Position ein primitives A steht und

$$\psi_{13}(k, i, A) = p_{k,i,A}$$

14. es einen Pfad in $U_{\mathcal{T}}(A_0)$ gibt, in dessen 0. Ebene A_0 steht,

$$\psi_{14} = q_{0,A_0}$$

15. sodass das lineare Konzept $\exists r_1 \cdot r_i.A$ *nicht* in $U_{\mathcal{T}}(A_0)$ auftaucht:

$$\psi_{15}(k, i, A) = \neg q_{i,A}$$

Zuletzt definieren wir nun die Formel $\varphi^{\mathcal{T}}_{A_0,B_0}$, die sich aus den oben definierten Formeln zusammensetzt.

$$\varphi^{\mathcal{T}}_{A_0,B_0} = \beta^{\mathcal{T}}_{B_0} \wedge missing^{\mathcal{T}}_{A_0,B_0}$$

mit

$$\beta^{\mathcal{T}}_{B_0} = repres^{\mathcal{T}}_{B_0} \wedge path^{\mathcal{T}}_{B_0}$$

Intuitiv gesprochen soll $\varphi^{\mathcal{T}}_{A_0,B_0}$ ausdrücken, dass es für ein lineares Konzept, das in $U_{\mathcal{T}}(B_0)$ vorkommt, kein Pfad in $U_{\mathcal{T}}(A_0)$ existiert, der dieses lineare Konzept als Blatt beinhaltet.

3.2.2 Korrektheitsbeweis der Reduktion

In diesen Kapitel wollen wir die Reduktionen zeigen, die grundlegend für die weitere Arbeit sein wird. Dazu wollen wir eine aussagenlogische Formel $\varphi^{\mathcal{T}}_{A_0,B_0}$ aus einem Konzept bilden, sodass gilt:

Theorem 3. *Für alle azyklischen TBoxen $\mathcal{T}$ und Konzeptnamen A_0, B_0 gilt:*

$\varphi^{\mathcal{T}}_{A_0,B_0}$ *ist erfüllbar* $\Leftrightarrow A_0 \not\sqsubseteq B_0$

Da aufgrund des Theorem 1 gilt:

$A_0 \not\sqsubseteq B_0 \Leftrightarrow leaves(U_{\mathcal{T}}(B_0)) \not\sqsubseteq leaves(U_{\mathcal{T}}(A_0))$

reicht es folgendes Lemma zu zeigen:

Lemma 4. *Für alle azyklischen TBoxen $\mathcal{T}$ und Konzeptnamen A_0, B_0 gilt:*

$leaves(U_{\mathcal{T}}(B_0)) \not\subseteq leaves(U_{\mathcal{T}}(A_0)) \Leftrightarrow \varphi^{\mathcal{T}}_{A_0,B_0}$ *erfüllbar.*

Beweis. $\Rightarrow$

Es gilt $leaves(U_{\mathcal{T}}(B_0)) \not\sqsubseteq leaves(U_{\mathcal{T}}(A_0))$ daher existiert ein lineares Konzept $F = \exists r_1 \ldots \exists r_n.C \in leaves(U_{\mathcal{T}}(B_0)) \setminus leaves(U_{\mathcal{T}}(A_0))$

Die Belegung V für die verschiedenen $p_{k,i,A}$ und $q_{i,B}$ Variablen bildet sich wie folgt:

Setze $V(p_{0,0,B_0}) = true$. Nun gehe in $U_{\mathcal{T}}(B_0)$ den Pfad $p = B_0 ... F$ von der Wurzel B_0 zum Blatt F ab. Gehe dabei für jeden durchlaufenen Knoten $\exists s_0 \ldots \exists s_m.D$ wie folgt vor: Gegeben sei i als bisher durchlaufene Knoten ab der Wurzel im Baum. Setze nun für jedes k mit $0 \leq k \leq m$ $V(p_{i,k,s_k}) = true$. Zudem setze jeweils $V(p_{i,k+1,D}) = true$.

Für die verschiedenen $q_{i,B}$ wird V wie folgt belegt: Setze zuerst $V(q_{0,A_0}) = true$. Nun gehe induktiv vor:

- Wenn gilt $V(j, A) = true$ mit $j \leq i$ und $A \equiv A_1 \sqcap A_2 \in \mathcal{T}$ dann: $V(q_{j,A_1}) = true$ und $V(q_{j,A_2}) = true$
- Wenn gilt $V(j, A) = true$ mit $j \leq i$ und $A \exists r.A_1 \in \mathcal{T}$ dann: $V(q_{j+1,A_1}) = true$

Im weiteren Verlauf gilt, dass alle nicht auf true gesetzen Variablen v gilt $V(v) = false$. Nun soll gezeigt werden, dass $V(\varphi^{\mathcal{T}}_{A_0,B_0}) = true$. Dazu zeigen wir das für alle Konjunkte von $\varphi^{\mathcal{T}}_{A_0,B_0}$ true sind. Zunächst betrachten wir

Zeige das V alle Konjunkte in $\varphi^{\mathcal{T}}_{A_0,B_0}$ erfüllt sind:

Zunächst betrachten wir die Konjunkte von $repres^{\mathcal{T}}_{B_0}$:

1. Ist erfüllt, da wir für jedes Ebene l und Position i immer nur ein $p_{l,i,S}$ (mit $S \in$) auf true setzen. Für alle anderen $p_{l,i,B}$ (mit $B \neq S$) gilt $V(p_{l,i,B}) = false$ da wir sie nicht auf true gesetzt haben.
2. Nach Konstruktion von $U_{\mathcal{T}}(B_0)$ gilt, dass auf jeden Ebene die Knoten nur mit einen Konzept der Form $\exists r_1 \ldots r_n.D$ (mit $n \geq 0$) beschriftet sein können. Daher kann nach Konstruktion von V bei $V(p_{l,i,r}) = true$ mit $r \in$ nur gelten dass $V(p_{l,i+1,S}) = true$ mit $S \in N_C \cup N_R$. Daher ist dieses Konjunkt erfüllt.
3. Ähnlich wie im vorherigen Punkt ist dieses Konjunkt erfüllt, da wir nach Konstruktion $U_{\mathcal{T}}(B_0)$ die Rollenpräfixe nur "aufsteigend" erweitern, also sich bei den Abstieg im Baum höchstens Rollen zum Rollenpräfix hinzugefügt, aber nicht entfernt werden.
4. Wenn auf einem Ebene l and der Position i gilt, dass kein p true ist, dann kann auch auf der folgenden Position $i + 1$ kein p in V auf true gesetzt worden sein, da nach Konstruktion nur Werte auf true gesetzt werden, die sich entweder auf der 0. Stelle in einen beliebigen Ebene befinden, oder auf einem Konzept- oder Rollennamen folgen.
5. Wenn wir während der Konstruktion von V auf das Blatt F treffen, dann gehen wir keine Ebene weiter, und setzen somit kein p mit größeren l auf true. Daher setzen wir wenn wir auf der 0. Stelle kein p erfüllt haben, auch kein p an einen späteren Stelle ($i \geq 1$). Daher ist dieses Konjunkt erfüllt.
6. Wenn wir während der Konstruktion von V auf das Blatt F treffen, dann gehen wir keine Ebene weiter, und setzen somit kein p mit größeren l auf true. Daher ist dieses Konjunkt erfüllt.
7. Auch für dieses Konjunkt kann man wie in den vorherigen beiden Konjunkten argumentieren.

Da alle Konjunkte erfüllt sind, gilt auch $V(repres^{\mathcal{T}}_{B_0}) = true$.

Nun zeigen wir, dass $V(path^{\mathcal{T}}_{B_0}) = true$:

8. Dieses Konjunkt gilt offensichtlich, da wir $V(p_{0,0,B_0}) = true$ setzen.
9. Während der Konstruktion von V entlang eines Pfades in $U_{\mathcal{T}}(B_0)$ gilt, dass wenn wir auf einer Ebene $V(p_{l,i,C}) = true$ setzen, für das $C \equiv C_0 \sqcap C_1 \in \mathcal{T}$ gilt, wir im Pfad auf der nächsten Ebene entweder ein Knoten finden mit C_0 oder C_1 als suffix. Daher gilt entweder $V(p_{l+1,i,C_0}) = true$ oder $V(p_{l+1,i,C_1}) = true$. Damit ist dieses

Konjunkt erfüllt.

10. Während der Konstruktion von V entlang eines Pfades in $U_{\mathcal{T}}(B_0)$ gilt, dass wenn wir auf einer Ebene $V(p_{l,i,C}) = true$ setzen, für das $C \equiv \exists r.C_1 \in \mathcal{T}$ gilt, wir im Pfad auf der nächsten Ebene ein Knoten dessen Beschriftung dem von vorherigen Knoten entspricht, wobei C durch $\exists r.C_1$ als suffix ersetzt. Daher gilt $V(p_{l+1,i,r}) = true$ und $V(p_{l+1,i+1,C_1}) = true$. Damit ist dieses Konjunkt erfüllt.

Im letzten Schritt muss gezeigt werden, dass $V(missing^{\mathcal{T}}_{A_0,B_0}) = true$ gitl.

11. Nach Konstruktion gilt offensichtlich dass wenn $V(q_{j,B}) = true$ mit $B \equiv B_1 \sqcap B_2$ auch $V(q_{j,B_1}) = true$ und $V(q_{j,B_2}) = true$ gilt.
12. Auch dieses Konjunkt ist erfüllt. Dabei kann man ähnlich wie beim vorherigen Konjunkt argumentieren.
13. Dieses Konjunkt gilt, da aufgrund der Belegungen aller p durch die gegebene Konstruktion mindestens ein $p_{k,i,A}$ mit A primitiv auf true gesetzt (da es in $U_{\mathcal{T}}(B_0)$ nur endliche Pfade gibts)
14. Nach Konstruktion gilt $V(q_{0,A_0}) = true$
15. Da wir die $p_{k,i,S}$ entlang eines Pfades zu $\exists r_1. \ldots \exists r_n.C \notin leaves(U_{\mathcal{T}}(A_0))$ gebildet haben, kommt das einzig wahre $p_{k,n,C}$ mit C primitiv nicht in $leaves(U_{\mathcal{T}}(A_0))$ vor. Daher gibt es kein Pfad in $U_{\mathcal{T}}(A_0)$ der zu dem Blatt $\exists r_1. \ldots \exists r_n.C$ führt. Daher gilt diese Konjukt.

Da auch $V(missing^{\mathcal{T}}_{A_0,B_0}) = true$ gilt, gilt

$$V(\varphi^{\mathcal{T}}_{A_0,B_0}) = V(repres^{\mathcal{T}}_{B_0} \wedge path^{\mathcal{T}}_{B_0} \wedge missing^{\mathcal{T}}_{A_0,B_0}) = true$$

$\Leftarrow$

$$\exists r_1. \ldots \exists r_n.A' \in leaves(U_{\mathcal{T}}(B_0))$$

$$\exists r_1. \ldots \exists r_n.A' \notin leaves(U_{\mathcal{T}}(A_0))$$

Gegeben sei, dass $V(\varphi^{\mathcal{T}}_{A_0,B_0}) = true$. Es zeigt sich, dass alle $p_{k,i,S}$ mit $V(p_{k,i,S}) = true$ einen Pfad in $U_{\mathcal{T}}(B_0)$ mit $\exists$-Prefix $\exists r_1 \ldots \exists r_n$ beschreiben: Für jedes k' mit $0 \leq k' \leq d_{\mathcal{T}}(B_0)$ gilt, dass die p_{k',i',B_0} den j. Knoten auf dem beschriebenen Pfad beschreiben. Nach den Konjunkten 1 - 6 beschreiben diese ein gültigen Konzept der Form $\exists r_1. \ldots \exists r_n.A'$ oder A'. Falls A' primitiv ist gibt es nach Konjunkt 7 nur $p_{l',i,B'}$ (mit $l < l' \leq d_{\mathcal{T}}(B_0)$ und $0 \leq i' \leq r_{\mathcal{T}}(B_0)$) mit $V(p_{l',i,B'}) = false$. Auch in $U_{\mathcal{T}}(B_0)$ haben wir das Ende des Pfades erreicht. Nach dem Konjunkt 13 muss das Literal $p_{0,0,B_0}$ erfüllt sein. Dies beschreibt die Wurzel von $U_{\mathcal{T}}(B_0)$. Durch die Konjunkte 11 und 12 muss nun aufsteigend ein Pfad durch V beschrieben werden, der als Blatt ein lineares Konzept der Form $\exists r_1 \ldots \exists r_i.C$ hat (mit C ein primitives Konzept).

Nach $missing^{\mathcal{T}}$ gilt, dass die $q_{i,A}$ mit $V(q_{i,A}) = true$ alle Pfade in $U_{\mathcal{T}}(A_0)$ mit einen Prefix von $\exists r_1 \ldots \exists r_n$ beinhaltet: Zunächst wird durch das Konjunkt 14 immer die Wurzel von $U_{\mathcal{T}}(A_0)$ beschrieben. Durch die Konjunkte 12 und 13 werden nun alle Pfade mit den Rollenpräfix $\exists r_1 \ldots \exists r_i$ beschrieben.

Durch das Konjunkt 15 $\neg q_{i,C}$ von $missing^{\mathcal{T}}_{A_0,B_0}$ gilt, dass das Blatt des vorherigen Pfades in $U_{\mathcal{T}}(B_0)$ in keinen Pfad von $U_{\mathcal{T}}(A_0)$ mit einen Prefix von $\exists r_1 \ldots \exists r_i$ als Blatt vorkommen darf.

Für das Konzept $\exists r_1 \ldots \exists r_i.C$ gilt also, dass es in $leaves(U_{\mathcal{T}}(B_0))$ aber nicht in $leaves(U_{\mathcal{T}}(A_0))$ vorkommt. Daher gilt

$$leaves(U_{\mathcal{T}}(B_0)) \not\subseteq leaves(U_{\mathcal{T}}(A_0))$$

□

3.2.3 Bilden der konjunktiven Normalform

Um die gebildeten Formeln direkt als Eingabe für die SAT-Solver nutzen zu können, müssen diese zunächst in die konjunktive Normalform (CNF) gebracht werden. Die CNF für aussagenlogische Formeln ist eine Konjunktion von Disjunktion von Literalen, also der Form:

$$(l_{01} \vee l_{02} \ldots l_{0n_0}) \wedge (l_{11} \vee l_{12} \ldots l_{1n_1}) \wedge \ldots \wedge (l_{k1} \vee l_{k2} \ldots l_{kn_k})$$

(mit $k, n \in \mathbb{N}$ und l_{ij} ein Literal)

Zum Bilden der konjunktiven Normalform der in den vorherigen Kapitel konstruierten Formeln werden folgende bekannte Äquivalenzen aus der Aussagenlogik verwendet:

- Definition der aussagenlogischen Implikation:

$$a \rightarrow b \equiv \neg a \vee b$$

- De Morgansche Gesetze:

$$\neg(a \wedge b) \equiv \neg a \vee \neg b$$
$$\neg(a \vee b) \equiv \neg a \wedge \neg b$$

- Distributivgesetz:

$$a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c)$$
$$a \wedge (b \vee c) \equiv (a \wedge b) \vee (a \wedge c)$$

- Assoziativität von Disjunktion:

$$a \vee (b \vee c) \equiv (a \vee b) \vee c$$

Man sieht leicht, dass unsere bisherige gebildete Formel $\varphi^{\mathcal{T}}_{A_0,B_0}$ bis auf die Konjunkte von $missing^{\mathcal{T}}_{A_0,B_0}$ beinahe in der konjunktiven Normalform sind. Im folgenden gehen wir für jedes Konjunkt durch und bilden jeweils eine äquivalente Formel in konjunktiver Normalform:

1.

$$\begin{aligned}\psi_1 &= \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{S\in N_C\cup N_R} \left(p_{\ell,i,S} \rightarrow \bigwedge_{B\neq S\in N_C\cup N_R} \neg p_{\ell,i,B} \right) \\ &\equiv \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{S\in N_C\cup N_R} \left(\neg p_{\ell,i,S} \vee \bigwedge_{B\neq S\in N_C\cup N_R} \neg p_{\ell,i,B} \right) \\ &\equiv \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{S\in N_C\cup N_R} \bigwedge_{B\neq S\in N_C\cup N_R} (\neg p_{\ell,i,S} \vee \neg p_{\ell,i,B}) \\ &=: \psi_1^{CNF}\end{aligned} \tag{6}$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite Äquivalenz gilt aufgrund des Distributivgesetzes.

2.

$$\begin{aligned}\psi_2 &= \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{r\in N_R} \left(p_{\ell,i,r} \rightarrow \bigvee_{A\in N_C} p_{\ell,i+1,A} \vee \bigvee_{r'\in N_R} p_{\ell,i+1,r'} \right) \\ &\equiv \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{r\in N_R} \left(\neg p_{\ell,i,r} \vee \bigvee_{A\in N_C} p_{\ell,i+1,A} \vee \bigvee_{r'\in N_R} p_{\ell,i+1,r'} \right) \\ &=: \psi_2^{CNF}\end{aligned} \tag{7}$$

Diese Äquivalenz gilt offensichtlich aufgrund der Definition der Implikation.

3.

$$\begin{aligned}\psi_3 &= \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{r\in N_R} \left(p_{\ell,i,r} \rightarrow (p_{\ell+1,i,r} \vee \bigwedge_{S\in N_C\cup N_R} \neg p_{\ell+1,i,s}) \right) \\ &\equiv \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{r\in N_R} \left(\neg p_{\ell,i,r} \vee (p_{\ell+1,i,r} \vee \bigwedge_{S\in N_C\cup N_R} \neg p_{\ell+1,i,s}) \right) \\ &\equiv \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{r\in N_R} \bigwedge_{S\in N_C\cup N_R} (\neg p_{\ell,i,r} \vee p_{\ell+1,i,r} \vee \neg p_{\ell+1,i,s}) \\ &=: \psi_3^{CNF}\end{aligned} \tag{8}$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite Äquivalenz gilt aufgrund des Distributivgesetzes.

4.

$$\begin{aligned}\psi_4 &= \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{A\in N_C} \left(p_{\ell,i,A} \rightarrow (\bigwedge_{A'\in N_C} \neg p_{\ell,i+1,A'} \wedge \bigwedge_{r'\in N_R} \neg p_{\ell,i+1,r'}) \right) \\ &\equiv \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{A\in N_C} \left(\neg p_{\ell,i,A} \vee (\bigwedge_{A'\in N_C} \neg p_{\ell,i+1,A'} \wedge \bigwedge_{r'\in N_R} \neg p_{\ell,i+1,r'}) \right) \\ &\equiv \bigwedge_{\ell\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \bigwedge_{A\in N_C} \left((\bigwedge_{A'\in N_C} \neg p_{\ell,i,A} \vee \neg p_{\ell,i+1,A'}) \wedge (\bigwedge_{r'\in N_R} \neg p_{\ell,i,A} \vee \neg p_{\ell,i+1,r'}) \right) \\ &=: \psi_4^{CNF}\end{aligned} \tag{9}$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite Äquivalenz gilt aufgrund des Distributivgesetzes.

5.

$$
\begin{aligned}
\psi_5 &= \bigwedge_{l\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \left(\left(\bigwedge_{A\in N_C} \neg p_{l,i,A} \wedge \bigwedge_{r\in N_R} \neg p_{\ell,i,r} \right) \rightarrow \left(\bigwedge_{A'\in N_C} \neg p_{\ell,i+1,A'} \wedge \bigwedge_{r'\in N_R} \neg p_{\ell,i+1,r'} \right) \right) \\
&\equiv \bigwedge_{l\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \left(\neg \left(\bigwedge_{A\in N_C} \neg p_{l,i,A} \wedge \bigwedge_{r\in N_R} \neg p_{\ell,i,r} \right) \vee \left(\bigwedge_{A'\in N_C} \neg p_{\ell,i+1,A'} \wedge \bigwedge_{r'\in N_R} \neg p_{\ell,i+1,r'} \right) \right) \\
&\equiv \bigwedge_{l\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \left(\left(\bigvee_{A\in N_C} \neg p_{l,i,A} \vee \bigvee_{r\in N_R} \neg p_{\ell,i,r} \right) \vee \left(\bigwedge_{A'\in N_C} \neg p_{\ell,i+1,A'} \wedge \bigwedge_{r'\in N_R} \neg p_{\ell,i+1,r'} \right) \right) \\
&\equiv \bigwedge_{l\leq d_{\mathcal{T}}(B_0), i\leq r_{\mathcal{T}}(B_0)} \left(\bigwedge_{A'\in N_C} \left(\bigvee_{A\in N_C} \neg p_{l,i,A} \vee \bigvee_{r\in N_R} \neg p_{\ell,i,r} \vee \neg p_{\ell,i+1,A'} \right) \wedge \right. \\
&\quad \left. \bigwedge_{r'\in N_R} \left(\bigvee_{A\in N_C} \neg p_{l,i,A} \vee \bigvee_{r\in N_R} \neg p_{\ell,i,r} \vee \neg p_{\ell,i+1,r'} \right) \right) \\
&=: \psi_5^{CNF}
\end{aligned}
\tag{10}
$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite Äquivalenz gilt aufgrund des De Morganschen Gesetzes und die dritte Äquivalenz gilt aufgrund des Distributivgesetzes.

6.

$$
\begin{aligned}
\psi_6 &= \bigwedge_{l\leq d_{\mathcal{T}}(B_0)} \left(\left(\bigwedge_{A\in N_C} \neg p_{\ell,0,A} \wedge \bigwedge_{r\in N_R} \neg p_{\ell,0,r} \right) \rightarrow \left(\bigwedge_{A'\in N_C} \neg p_{\ell+1,0,A'} \wedge \bigwedge_{r'\in N_R} \neg p_{\ell+1,0,r'} \right) \right) \\
&\equiv \bigwedge_{l\leq d_{\mathcal{T}}(B_0)} \left(\neg \left(\bigwedge_{A\in N_C} \neg p_{\ell,0,A} \wedge \bigwedge_{r\in N_R} \neg p_{\ell,0,r} \right) \vee \left(\bigwedge_{A'\in N_C} \neg p_{\ell+1,0,A'} \wedge \bigwedge_{r'\in N_R} \neg p_{\ell+1,0,r'} \right) \right) \\
&\equiv \bigwedge_{l\leq d_{\mathcal{T}}(B_0)} \left(\left(\bigvee_{A\in N_C} p_{\ell,0,A} \vee \bigvee_{r\in N_R} p_{\ell,0,r} \right) \vee \left(\bigwedge_{A'\in N_C} \neg p_{\ell+1,0,A'} \wedge \bigwedge_{r'\in N_R} \neg p_{\ell+1,0,r'} \right) \right) \\
&\equiv \bigwedge_{l\leq d_{\mathcal{T}}(B_0)} \left(\bigwedge_{A'\in N_C} \left(\bigvee_{A\in N_C} p_{\ell,0,A} \vee \bigvee_{r\in N_R} p_{\ell,0,r} \vee \neg p_{\ell+1,0,A'} \right) \wedge \right. \\
&\quad \left. \bigwedge_{r'\in N_R} \left(\bigvee_{A\in N_C} p_{\ell,0,A} \vee \bigvee_{r\in N_R} p_{\ell,0,r} \vee \neg p_{\ell+1,0,r'} \right) \right) \\
&=: \psi_6^{CNF}
\end{aligned}
\tag{11}
$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite Äquivalenz gilt aufgrund des De Morganschen Gesetzes und die dritte Äquivalenz gilt aufgrund des Distributivgesetzes.

7.

$$\begin{aligned}
\psi_7 &= \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{A \in N_C \text{ mit } A \text{ primitiv}} \left(p_{\ell,i,A} \rightarrow \left(\bigwedge_{A' \in N_C} \neg p_{\ell+1,0,A'} \wedge \bigwedge_{r' \in N_R} \neg p_{\ell+1,0,r'} \right) \right) \\
&\equiv \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{A \in N_C \text{ mit } A \text{ primitiv}} \left(p_{\ell,i,A} \rightarrow \left(\bigwedge_{A' \in N_C} \neg p_{\ell+1,0,A'} \wedge \bigwedge_{r' \in N_R} \neg p_{\ell+1,0,r'} \right) \right) \\
&\equiv \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{A \in N_C \text{ mit } A \text{ primitiv}} \left(\bigwedge_{A' \in N_C} (\neg p_{\ell,i,A} \vee \neg p_{\ell+1,0,A'}) \wedge \bigwedge_{r' \in N_R} (\neg p_{\ell,i,A} \vee \neg p_{\ell+1,0,r'}) \right) \\
&=: \psi_7^{CNF}
\end{aligned} \tag{12}$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite Äquivalenz gilt aufgrund des Distributivgesetzes.

8.

$$\psi_8 = p_{0,0,B_0} = \psi_8^{CNF}$$

Diese Teilformel ist bereits in der konjunktiven Normalform.

9.

$$\begin{aligned}
\psi_9 &= \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{C \equiv C_1 \sqcap C_2 \in \mathcal{T}} (p_{\ell,i,C} \rightarrow p_{\ell+1,i,C_1} \vee p_{\ell+1,i,C_2}) \\
&\equiv \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{C \equiv C_1 \sqcap C_2 \in \mathcal{T}} (\neg p_{\ell,i,C} \vee p_{\ell+1,i,C_1} \vee p_{\ell+1,i,C_2}) \\
&=: \psi_9^{CNF}
\end{aligned} \tag{13}$$

Diese Äquivalenz ist aufgrund der Definition der Implikation gültig.

10.

$$\begin{aligned}
\psi_{10} &= \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{C \equiv \exists r.C' \in \mathcal{T}} (p_{\ell,i,C} \rightarrow p_{\ell+1,i,r} \wedge p_{\ell+1,i+1,C'}) \\
&\equiv \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{C \equiv \exists r.C' \in \mathcal{T}} (\neg p_{\ell,i,C} \vee (p_{\ell+1,i,r} \wedge p_{\ell+1,i+1,C'})) \\
&\equiv \bigwedge_{\ell \leq d_{\mathcal{T}}(B_0), i \leq r_{\mathcal{T}}(B_0)} \bigwedge_{C \equiv \exists r.C' \in \mathcal{T}} ((\neg p_{\ell,i,C} \vee p_{\ell+1,i,r}) \wedge (\neg p_{\ell,i,C} \vee p_{\ell+1,i+1,C'})) \\
&=: \psi_{10}^{CNF}
\end{aligned} \tag{14}$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite gilt aufgrund des Distributivgesetzes.

Für die Konjunkte $\psi_1^{CNF}, \ldots, \psi_1 0^{CNF}$ sieht man nun leicht, dass diese in konjunktiver Normalform sind. Für die ersten beiden Konjunkte von $missing_{A_0,B_0}^{\mathcal{T}}$ kann man ähnlich vorgehen:

11.

$$\begin{aligned}\psi_{11}(k,i,A)(k,i,A) &= \bigwedge_{B\equiv B_1\sqcap B_2\in\mathcal{T}}\bigwedge_{j\leq i}(q_{j,B}\rightarrow q_{j,B_1}\wedge q_{j,B_2})\\ &\equiv \bigwedge_{B\equiv B_1\sqcap B_2\in\mathcal{T}}\bigwedge_{j\leq i}(\neg q_{j,B}\vee(q_{j,B_1}\wedge q_{j,B_2}))\\ &\equiv \bigwedge_{B\equiv B_1\sqcap B_2\in\mathcal{T}}\bigwedge_{j\leq i}((\neg q_{j,B}\vee q_{j,B_1})\wedge(\neg q_{j,B}\vee q_{j,B_2}))\\ &=: \psi_{11}(k,i,A)^{CNF}\end{aligned} \tag{15}$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite gilt aufgrund des Distributivgesetzes.

12.

$$\begin{aligned}\psi_{12}(k,i,A) &= \bigwedge_{B\equiv\exists r.B'\in\mathcal{T}}\bigwedge_{j\leq i}(p_{k,j,r}\wedge q_{j,B}\rightarrow q_{j+1,B'})\\ &\equiv \bigwedge_{B\equiv\exists r.B'\in\mathcal{T}}\bigwedge_{j\leq i}(\neg(p_{k,j,r}\wedge q_{j,B})\vee q_{j+1,B'})\\ &\equiv \bigwedge_{B\equiv\exists r.B'\in\mathcal{T}}\bigwedge_{j\leq i}(\neg p_{k,j,r}\vee\neg q_{j,B}\vee q_{j+1,B'})\\ &=: \psi_{12}(k,i,A)^{CNF}\end{aligned} \tag{16}$$

Die erste Äquivalenz gilt aufgrund der Definition der Implikation. Die zweite Äquivalenz gilt aufgrund des De Morganschen Gesetzes.

Nun muss noch das letzte in $missing^{\mathcal{T}}_{A_0,B_0}$ dargestellte Konjunkt in die konjunktive Normalform gebracht werden. Dieses Konjunkt nennen wir $missRest^{\mathcal{T}}_{A_0,B_0}$. Diese sieht wie folgt aus:

$$missRest^{\mathcal{T}}_{A_0,B_0} = \bigvee_{A\in N_C\ mit\ A\ primitiv}(\psi_{13}(k,i,A)\psi_{14}\wedge\psi_{15}(k,i,A))$$

Für dieses Konjunkt können wir nicht mehr einfache Äquivalenzregeln anwenden, da, aufgrund der äußeren Disjunktion, die entstandene Formel exponentiell größer als die Ausgangsformel werden könnte. Daher nutzen wir einen Algorithmus zur Übersetzung einer Formel in eine erfüllbarkeitsäquivalente CNF der in Plaisted and Greenbaum (1986, S. 294) vorgestellt wurde, und dessen resultierende Formel nicht exponentiell größer wirde (Plaisted and Greenbaum, 1986, vgl. S.298,Thm.3). Da wir im Theorem 3 auf Erfüllbarkeit reduziert haben, reicht die Erfüllbarkeitsäquivalenz für die weitere Arbeit in dieser Thesis aus.

Zuerst führen wir eine Bezeichnung für die Konjunkte $\psi_{13}(k,i,A), \psi_{14}, \psi_{15}(k,i,A)$ ein:

$$mconj(k,i,A) = \psi_{13}(k,i,A)\wedge\psi_{14}\wedge\psi_{15}(k,i,A)$$

Nun wollen wir den von Plaisted und Greenbaum beschriebenen Algorithmus schrittweise anwenden. In dem Algorithmus wird eine aussagenlogische Formel θ mithilfe der induktiven Anwendung von Regeln in eine eine erfüllbarkeitsäquivalente CNF übersetzt. Die

erfüllbarkeitsäquivalent Formel zu θ wäre dann $x_\theta \wedge (\theta)^+$ Die in dieser Arbeit verwendeten Regeln sind (unter Verwendung der in dieser Arbeit genutzten Notation):

$$(A \wedge B)^+ = (x_{A \wedge B} \rightarrow x_A \wedge x_B) \wedge A^+ \wedge B^+$$
$$(A \vee B)^+ = (x_{A \vee B} \rightarrow x_A \vee x_B) \wedge A^+ \wedge B^+$$

mit A und B aussagenlogischen Formeln und die jeweiligen x_w Variablen die so mit einer jeweiligen Teilformel verbunden wird, dass x_w erfüllbar ist, wenn die Teilformel erfüllbar ist. Weitere Regeln sind in der Arbeit von Plaisted und Greenbaum zu finden (Plaisted and Greenbaum, 1986, S. 294). In dieser Arbeit wende wir diese Regel auf große Konjunktionen und Disjunktionen an. Wir verwenden daher die Regeln in folgender Form:

$$(\bigwedge_i \theta_i)^+ = (x_{conj} \rightarrow \bigwedge_i x_{\theta_i}) \wedge \bigwedge_i (\theta_i)^+$$
$$(\bigvee_i \theta_i)^+ = (x_{conj} \rightarrow \bigvee_i x_{\theta_i}) \wedge \bigwedge_i (\theta_i)^+$$

Es zeigt sich, dass die Anwendung dieser Regeln auch erfüllbarkeitsäquivalente Formeln ergeben, da wir sonst die Disjunktion/Konjunktionen verschachtelt definieren müssten. Als Variablen verwenden wir hier x_miss, die erfüllbarkeitsäquivalent zur Gesamtformel $missRestg^{MT}_{A_0,B_0}$ sein soll, und verschiendene Variablen $x_{mconj(k,i,A)}$ die jeweils zur Formel $mconj(k,i,A)$ erfüllbarkeitsäquivalent sein sollen. Im folgenden wird jeweils zuerst die Regel angewendet, und danach die entstandenen Formeln mit den bekannten Äquivalenzen in die konjunktive Normalform übersetzt, sodass die entstehende Formel $x_{miss} \wedge (missRestg^{MT}_{A_0,B_0})^+$ bereits in konjunktiver Normalform ist:

1.

$$\begin{aligned}(missRestg^{MT}_{A_0,B_0})^+ = &\left(x_{miss} \rightarrow \left(\bigvee_{k \leq d_\mathcal{T}(B_0), i \leq r_\mathcal{T}(B_0), Aprim.} x_{mconj(k,i,A)}\right)\right) \\ &\wedge \bigwedge_{k \leq d_\mathcal{T}(B_0), i \leq r_\mathcal{T}(B_0), Aprim.} (mconj(k,i,A))^+ \\ \equiv &\left(\neg x_{miss} \vee \bigvee_{k \leq d_\mathcal{T}(B_0), i \leq r_\mathcal{T}(B_0), Aprim.} x_{mconj(k,i,A)}\right) \\ &\wedge \bigwedge_{k \leq d_\mathcal{T}(B_0), i \leq r_\mathcal{T}(B_0), Aprim.} (mconj(k,i,A))^+\end{aligned} \tag{17}$$

Im ersten Schritt wird die Formel nach den Algorithmus von Plaisted und Greenbaum in eine äquivalente Formel übersetzt. Die darauffolgende Äquivalenz gilt aufgrund der Definition der Implikation.

2.

$$\begin{aligned}(mconj(k,i,A))^+ = &(x_{mconj(k,i,A)} \rightarrow (p_{k,i,A} \wedge q_{0,A_0} \wedge \neg q_{i,A})) \\ \equiv &(\neg x_{mconj(k,i,A)} \vee (p_{k,i,A} \wedge q_{0,A_0} \wedge \neg q_{i,A})) \\ \equiv &(\neg x_{mconj(k,i,A)} \vee p_{k,i,A}) \wedge (\neg x_{mconj(k,i,A)} \vee q_{0,A_0}) \\ &\wedge (\neg x_{mconj(k,i,A)} \vee \neg q_{i,A})\end{aligned} \tag{18}$$

Im ersten Schritt wird die Formel nach den Algorithmus von Plaisted und Greenbaum in eine äquivalente Formel übersetzt. Die darauffolgende Äquivalenzen gelten aufgrund der Definition der Implikation und des Distributivgesetzes.

Die erfüllbarkeitsäquivalente CNF von $missing^{\mathcal{T}}_{A_0,B_0}$ setzt sich daher wie folgt zusammen:

$$missing^{\mathcal{T},CNF}_{A_0,B_0} = \psi_{11}(k,i,A)^{CNF} \wedge \psi_{12}(k,i,A)^{CNF} \wedge (missRest^{\mathcal{T}}_{A_0,B_0})^{+}$$

Da $\psi_{11}(k,i,A)^{CNF}$ und $\psi_{12}(k,i,A)^{CNF}$ äquivalent und $(missRest^{\mathcal{T}}_{A_0,B_0})^{+}$ erfüllbarkeitsäquivalent (Plaisted and Greenbaum, 1986, vgl. Theorem 2) zu $missing^{\mathcal{T}}_{A_0,B_0}$ sind, muss auch $missing^{\mathcal{T},CNF}_{A_0,B_0}$ erfüllbarkeitsäquivalent zu $missing^{\mathcal{T}}_{A_0,B_0}$ sein.

Wenn wir jetzt die Formel $\varphi^{\mathcal{T},CNF}_{A_0,B_0}$ bilden, sieht man leicht dass diese auch in der konjunktiven Normalform sind, da die Gesamteformel immer noch den Aufbau der Normalform folgt:

$$\varphi^{\mathcal{T},CNF}_{A_0,B_0} = \psi_1^{CNF} \wedge \psi_2^{CNF} \wedge \ldots \wedge \psi_{10}^{CNF} \wedge missing^{\mathcal{T},CNF}_{A_0,B_0}$$

Für die ersten zwölf Formeln, die durch Anwenden einfacher Äquivalenzen gebildet wurden, zeigt es sich, dass die konjunktive Normalform in polynomieller Zeit berechnet wird, da jeder beschriebene Schritt in polynomieller Zeit durchgeführt werden kann und es nur endlich viele Schritte gibt. Auch ist die Länge der Formeln polynomiell beschränkt, da die einzige Regel, die die Größe der Formel verändert, das Distributivgesetz ist. Diese wird für all diese Formeln nur einmal angewendet und vergrößert die Formel maximal um das Doppelte der Länge der Ausgangsformel. Ähnliches gilt für die Formeln, die nach den Algorithmus von Plaisted und Greenbaum gebildet wurden, für die in der Arbeit von Plaisted und Greenbaum gezeigt wurde, dass sie in polynomieller Zeit berechnet werden können und polynomiell in ihrem Platz beschränkt sind (Plaisted and Greenbaum, 1986, vgl. Theorem 3, S. 298).

Offenbar ist $\varphi^{\mathcal{T},CNF}_{A_0,B_0}$ auch erfüllbarkeitsäquivalent zu $\varphi^{\mathcal{T}}_{A_0,B_0}$, da alle Konjunkte äquivalent oder erfüllbarkeitsäquivalent zu ihren entsprechenden Konjunkten in $\varphi^{\mathcal{T}}_{A_0,B_0}$ definiert wurden. Daher gilt:

Theorem 5. *Für alle azyklischen TBoxen $\mathcal{T}$ und Konzeptnamen A_0, B_0 gilt:*
$\varphi^{\mathcal{T},CNF}_{A_0,B_0}$ *ist erfüllbar* $\Leftrightarrow A_0 \not\sqsubseteq B_0$

3.2.4 Algorithmus

In diesen Abschnitt wollen wir nun den Algorithmus beschreiben, der die in den vorherigen Abschnitten beschriebene Reduktion nutzt um das das Subsumtionsproblem für eine gegebene Ontologie und einen

Eingabe: Ontologie $\mathcal{T}$, Konzeptnamen A_0 und B_0

Vorgehen: Zuerst konstruieren wir aus den Eingaben A_0 und B_0 die Formel $\varphi^{\mathcal{T},CNF}_{A_0,B_0}$. Nun soll getestet werden ob diese Formel erfüllbar ist. Wenn die Formel *nicht* erfüllbar ist, gilt $\mathcal{T} \models A_0 \sqsubseteq B_0$, sonst gilt $\mathcal{T} \models A_0 \not\sqsubseteq B_0$

Ausgabe: Gilt $\mathcal{T} \models A_0 \sqsubseteq B_0$

4 Implementierung

Nachdem wir in den bisherigen Kapiteln die theoretischen Grundlagen der Entwicklung des Reasoner für $\mathcal{EL}^{\mathcal{F}}$ behandelt haben, wollen wir im folgenden Kapitel die Implementierung dieses Reasoner beschreiben. Als Namen für den von uns entwickelten Reasoner wurde "Funky" gewählt.

Die grundlegende Idee des in dieser Thesis implementierten Reasoner ist, dass eine $\mathcal{EL}^{\mathcal{F}}$ Ontologie $\mathcal{T}$ aus einer OWL-Ontologie eingelesen wird, um das Subsumtionproblem für zwei gegebene Konzepte A_0 und B_0 für diese Ontologie zu entscheiden (also $\mathcal{T} \models A_0 \sqsubseteq B_0$). Der Reasoner hält sich dabei an den im 3.2.4 Kapitel beschriebenen Algorithmus.

4.1 Architektur

Als Eingabe erhält Funky eine azyklische Ontologie $\mathcal{T}$ in $\mathcal{EL}^{\mathcal{F}}$, die sich nicht notwendig in Normalform befinden muss, und zwei Konzeptnamen A_0 und B_0 aus der Ontologie.

Grundlegend setzt sich die Implementierung von Funky aus 3 Modulen zusammen:

1. **funky-OntologyManager**: Dieses Modul ist dafür zuständig die gegebene OWL-Ontologie einzulesen, den Normalisierer aufzurufen, und die normalisierte Ontologie zu verwalten. Dabei speichert der OntologyManager zudem die Menge der Axiome und Konzepte und stellt diese zur Verfügung.
2. **funky-SimpleNormalizer**: In diesen Modul findet die Normalisierung der gegebenen Ontologie statt. Dabei werden die Normalisierungsregeln nach Haase (2007) verwendet.
3. **funky-SubsumtionDecider**: Das Modul SubsumtionDecider ist dafür zuständig, zu einer gegebenen (normalisierten) Ontologie die Subsumtion für zwei gegebene Konzeptnamen, mit der im ersten Teil der Arbeit beschriebenen Reduktion, zu entscheiden. Hier werden die Konzept- und Rollentiefen, sowie die von einem Konzeptnamen abhängigen Konzepte als Map verwaltet. Die Einträge in den Maps für die Konzeptnamen werden berechnet wenn sie benötigt werden. Dabei wird auf ein Sat-Solver aus der Kodkod-API zurückgegriffen.

Die Module und ihre Abhängigkeit untereinander wird in der Abbildung 1 dargestellt. Man sieht aus, welche Bibliotheken genutzt werden. Es zeigt sich, dass der Subsumtion-Decider von den anderen Modulen abhängig ist, da es diese benötigt, um die Ontologie zu laden, und zu normalisieren. Zudem benutzt Subsumtiondecider die KodKod API um SAT Solver aus diesen zu nutzen. Die OWL-API und jcel werden vom OntologyManager genutzt um OWL Ontologien zu laden und zu verwalten.

4.2 Designentscheidungen

In diesem Kapitel wollen wir verschiedene Designentscheidungen für das vorliegende System beschreiben und begründen.

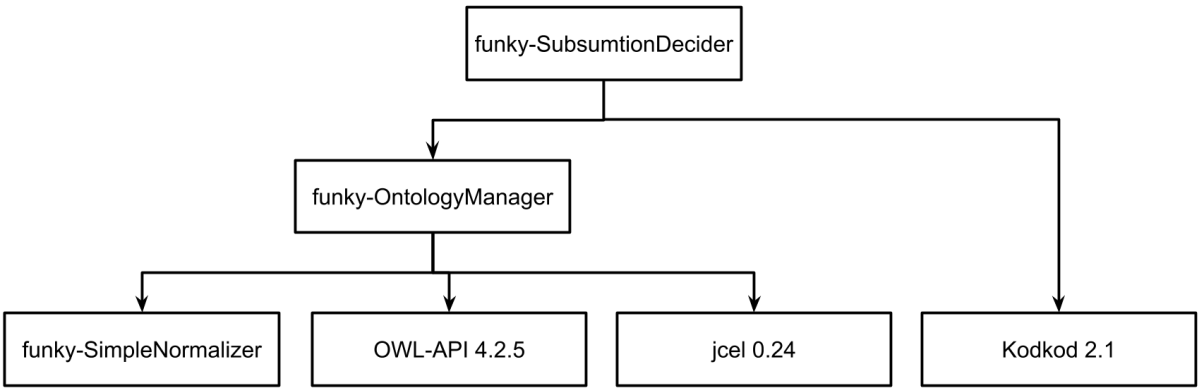


Abbildung 1: Module und Verknüpfungen des Systems

OWL-API

Um mit Ontologien zu arbeiten hat sich wie bereits im Kapitel 2. beschrieben, die Ontologie-Sprache OWL herausgebildet. Aufgrund ihrer Verbreitung und der Existenz vieler Beispielsontologien haben wir daher in dieser Arbeit ein Reasoner implementiert, der die in Horridge and Bechhofer (2011) beschriebene OWL-API nutzt, um über OWL-Ontologien Reasoning betreiben zu können.

jcel

jcel ist die Implementierung des in Suntisrivaraporn (2009) beschriebenen Algorithmus zur Klassifikation einer Ontologie von Mendez (2007) in Java. Diese wurde zur Implementation in dieser Arbeit genutzt um die aus der Ontologie (mit der Ausdruckstärke $\mathcal{ELHI}f_{\mathcal{R}^+}$) eingelesenen Konzepte leichter zu verwalten. Die Konzepte werden in jcel von der Klasse IntegerEntityManager als Integer verwaltet [1], was unter anderem auch einen Vorteil in der Speichernutzung bringen soll (Mendez, 2007, vgl. S.33). Zudem ist die Implementation der Normalisierung in ihrer Form angelehnt an die Normalisierung in jcel, wobei andere Normalisierungsregeln genutzt werden.

Wahl des SAT-Solver / API

Bei der Wahl der SAT-Solver wurde hauptsächlich darauf geachtet das die SAT-Solver in Vergleichen wie der als performant erscheinen. Dabei sollte das einbinden und Wechseln der Solver relativ leicht geschehen.

Dazu wurde die Constraint-Solver Bibliothek Kodkod von Torlak and Jackson (2007) genutzt [2]. Die SAT-Solver Bibliothek ist dabei nur ein Teil des gesamten Kodkod-System, kann aber eigenständig genutzt werden. [3].

Diese Bibliothek wurde gewählt, da sie einerseits einen einfachen und Schnellen Wechseln der Solver ermöglichte (dies geschieht einfach durch den Aufruf an die Klasse der

[1] IntegerEntityManager: http://jcel.sourceforge.net/javadoc/de/tudresden/inf/lat/jcel/coreontology/datatype/IntegerE
[2] Kodkod: http://alloy.mit.edu/kodkod/
[3] Kodkod - SAT Solver Doc: http://alloy.mit.edu/kodkod/release/current/doc/kodkod/engine/satlab/SATSolver.html

SATFactory der Form SATFactory.MiniSat). Zudem sind in der Bibliothek verschiedenen SAT-Solver nutzbar, sodass diese verglichen werden können. Die in Kodkod nutzbaren SATSolver sind:

- **Sat4J**: Sat4J ist eine Implementierung des Algorithmus von Eén and Sörensson (2003) die anders als MiniSat, vollständig in Java implementiert wurde. Dies ermöglicht uns zu vergleichen, ob ein vollständig in Java implentierter Solver ein Vorteil gegenüber den nicht in Java implementierten Solvern hat. Leider ist Sat4J aber im Vergleich zu den anderen vorliegenden Solvern zumindest in verschiedenen Wettbewerben nicht so performant wie die anderen Solver.
- **MiniSat**: MiniSat wurde von Eén and Sörensson (2003) entwickelt und implementiert. Genutzt wird MiniSat 2, das z.B. den SAT-Race von 2008 gewonnen hat [4]
- **Glucose**: Glucose ist eine auf Minisat basierender SAT-Solver der zuerst in Audemard and Simon (2009) vorgestellt wurde, und in den letzten Jahren regelmäßig erfolgreich bei den "The international SAT Competition" abschnitt [5].
- **Lingeling**: Lingeling ist ein jüngerer SAT-Solver entwickelt und implementiert von Biere (2010). Dieser SAT-Solver hat auch in jüngerer Zeit größere Erfolge bei "The international SAT Competition" erzielt und 2014 sogar gewonnen (SAT+UNSAT ohne Parallelität)[6].
- **Plingeling**: Plingeling ist eine weitere Implementierung von Biere (2010), die eine Nebenläufige version von Lingeling darstellt. Auch diese Version hat in den "The international SAT Competition" in den letzten Jahren gewonnen (2014, SAT+UNSAT mit Parallelität).

Ein weiterer Grund der für Kodkod spricht ist, dass vorkompilierte native Bibliotheken für verschiedene Betriebssysteme für die Solver direkt heruntergeladen werden können [7].

4.3 Optimierungen

Im Laufe der Implementierung wurden mehrere Optimierungen durchgeführt, und verglichen. In diesem Kapitel wollen wir eben diese diskutieren und bewerten. Anzumerken ist, dass wir für die hier diskutierten Optimierungen nicht immer das Optimun erreicht haben müssen, und weitere Optimierungen, auch in den hier genannten Bereichen sinnvoll sein können.

Einschränkung der Eingabegrößen

Die effizienteste Optimierung während der Implementierung war die Einschränkung der Eingabegrößen. Die einzelnen Teilformeln müssen z.B.: nicht über alle Konzeptnamen, sondern nur über die von den in der Subsumtion betroffenen Konzepte, gebildet werden.

Dabei werden für die einzelnen Teilformeln für die Anfrage ob A_0 von B_0 nicht subsumiert wird ($A_0 \not\sqsubseteq B_0$) die in Tabelle 2 beschriebenen Paramenter (die jeweiligen Tiefen

[4]SAT-Race 2008: http://baldur.iti.uka.de/sat-race-2008/results.html
[5]The international SAT Competition http://www.satcompetition.org/
[6]The international SAT Competition http://www.satcompetition.org/
[7]Pre-compiled native libraries: http://alloy.mit.edu/kodkod/download.html

entsprechen den Tiefen von B_0 im Bezug zur TBox $\mathcal{T}$) benötigt.

Tabelle 2: Komplexität mit verschiedenen TBoxen

Teilformel	benötigte Parameter
ψ_1	maximale Rollentiefe, maximale Konzepttiefe (depth), N_C, N_R
ψ_2	maximale Rollentiefe, maximale Konzepttiefe (depth), N_C, N_R
ψ_3	maximale Rollentiefe, maximale Konzepttiefe (depth), N_C, N_R
ψ_4	maximale Rollentiefe, maximale Konzepttiefe (depth), N_C, N_R
ψ_5	maximale Rollentiefe, maximale Konzepttiefe (depth), N_C, N_R
ψ_6	maximale Rollentiefe, maximale Konzepttiefe (depth), N_C, N_R
ψ_7	maximale Rollentiefe, maximale Konzepttiefe (depth), N_C, N_R
ψ_8	B_0
ψ_9	maximale Rollentiefe, Menge der Axiome der Form $A \equiv B \sqcap C$ aus $\mathcal{T}$ (mit $A, B, C \in N_C$)
ψ_{10}	maximale Rollentiefe,Menge der Axiome der Form $A \equiv \exists r.B$ aus $\mathcal{T}$ (mit $A, B \in N_C$ und $r \in N_R$)
ψ_{11}	maximale Rollentiefe, maximale Konzepttiefe (depth), Menge der Axiome der Form $A \equiv B \sqcap C$ aus $\mathcal{T}$ (mit $A, B, C \in N_C$)
ψ_{12}	maximale Rollentiefe, maximale Tiefe (depth), maximale Konzepttiefe (depth), Menger Axiome der Form $A \equiv \exists r.B$ aus $\mathcal{T}$ (mit $A, B \in N_C$ und $r \in N_R$)
$\psi_{13} \wedge \psi_{14} \wedge \psi_{15}$	A_0, maximale Rollentiefe, maximale Konzepttiefe (depth), Menge der primitiven Konzepte

Nun wollen wir diskutieren, wie man den Wertebereich dieser Parameter sinnvoll einschränken kann. Für die Formeln $\psi_1, \ldots, \psi_7$ sind folgende Einschränkungen sinnvoll:

- **maximale Rollentiefe:** Hier kann man die Rollentiefe (roledepth) von B_0 in $\mathcal{T}$ nutzen, da wir hier den Baum $U_{\mathcal{T}}(B_0)$ beschreiben. Aus den selben Grund kann dieser Parameter im Allgemeinen auch nicht kleiner gewählt werden, da sonst nicht alle Pfade in $U_{\mathcal{T}}(B_0)$ erfasst werden könnten.
- **maximale Konzepttiefe:** Hier kann man die Konzepttiefe (depth) von B_0 nutzen in $\mathcal{T}$ nutzen, und dabei ähnlich wie im vorherigen Punkt argumentieren.
- $\mathbf{N_C}$ **und** $\mathbf{N_R}$**:** Die Mengen der Konzepte und der Rollen kann man für diese Teilformeln auf die Konzepte/Rollen einschränken, die im Abhängigkeitsgraph von B_0 in Bezug zu $\mathcal{T}$ erreichbar/betroffen sind. Offenbar würden kleinere Menge dazu führen, dass wir den Graph $U_{\mathcal{T}}(B_0)$ nicht mehr vollständig beschreiben.

Die Formel ψ_8 braucht offenbar nur B_0 als Eingabe, und muss daher nicht eingeschränkt werden. Für die Teilformeln ψ_9 und ψ_{10} können die maximale Rollen- und Konzepttiefe so eingeschränkt werden wie für die vorherigen Teilformeln.

- **Menge der Axiome der Form** $A \equiv \exists r.B$ **aus** $\mathcal{T}$ **(mit** $A, B \in N_C$ **und** $r \in N_R$**):** Hier nutzen wir alle Axiome dieser Form, die beim bilden des Abhängigkeitsgraph von B_0 benutzt worden sind. Dieser Parameter sollte nicht kleiner gewählt werden, da sonst nicht alle Pfade in $U_{\mathcal{T}}(B_0)$ erfasst werden könnten.
- **Menge der Axiome der Form** $A \equiv B \sqcap C$ **aus** $\mathcal{T}$ **(mit** $A, B, C \in N_C$**):** Auch hier gehen wir wie im vorherigen Punkt vor.

Für $\psi_{11}, \ldots, \psi_{15}$ gilt, das die maximale Rollentiefe, maximale Konzepttiefe, die Menge der Konzepte (N_C) und auch die Axiome ähnlich wie für die vorherigen Teilformeln gewählt werden, nur in Abhängigkeit zu A_0 statt in Abhängigkeit zu B_0. Hierfür kann auch ähnlich wie für die Parameter der vorherigen Teilformeln argumentiert werden. Zudem kommt aber noch die Menge der primitiven Konzepte als Parameter hinzu:

- **Menge der primitiven Konzepte**: Diese Menge kann stark eingeschränkt werden, indem man nur die primitiven Konzepte nimmt, die im Abhängkeitsgraph von B_0 vorkommen. Dies reicht, da wir in den Formeln $\psi_{13}, \ldots, \psi_{15}$ Aussagen über Pfade mit primitiven Konzepten im Blatt treffen, die in $U_{\mathcal{T}}(B_0)$ vorkommen.

Optimierungen der aussagenlogischen Formel

Eine weitere wichtige Optimierung war die an aussagenlogischen Formeln. Diese Optimierungen sind in der vorliegenden Version bereits eingearbeitet.

Änderungen waren z.B.: die Formel, dass "auf jeden nichts nichts folgt" statt in einer großen Formel, in mehrere kleinere Formeln auszudrücken oder bereits vorhandene Teilformeln zu nutzen. Die altet Formel hatte die Aussage getroffen, dass wenn auf einem Level ℓ and der Stelle i kein p wahr ist, auf allen Leveln $\ell' \geq \ell$ und Stellen $i \geq i'$ auch kein p wahr sein kann. Diese Formel hatte aber offenbar zur Folge dass ein große Anzahl an Klauseln dem Solver übergeben werden müssen. Dies wird in der vorliegenden Version sparsamer durch die Formeln ψ_5 und ψ_6 ausgedrückt.

Besonders in diesen Punkt können dem Autor nicht aufgefallene Optimierungen noch möglich sein.

Weitere Optimierungen

Hier wollen wir weitere Optimierungen, die in der Implementierung durchgeführt worden sind aufzählen. Diese hatten jeweils eine weit geringeren Einfluss auf die Laufzeit.

- Einschränkung der Anzahl der Aussagenlogischen Variablen: Der von Kodkod gegebene Solver erwartet vor dem hinzufügen der Klauseln die Anzahl der aussagenlogischen Variablen. Dies muss mindestens die Zahl der wirklich verwendeten aussagenlogischen Variablen beschreiben, kann aber größer sein. Es hat sich während der Implementierung aber gezeigt, dass eine genaue Einschränkung notwendig ist, da eine zu grobe obere Schranke die Laufzeit des Solvers wesentlich verschlechtert.
- Neben den genannten Optimierungen wurden auch mehrere Java-spezifische Optimierungen genutzt, die aber keinen wesentlichen Laufzeit-Vorteil gebracht haben. Daher werden sie in dieser Arbeit auch nicht genauer beschrieben.

4.4 Verbesserungsvorschläge

Zur Implementierung unseres Systems haben wir Java genutzt, da die OWL-Api in Java als ausführlicher gilt, als in anderen Sprachen. Alternativ zu den von uns genutzten Implementierung in Java, könnte eine Implementierung in einer Sprache wie C/C++ sinnvoll sein. Die OWL-API für andere Sprachen ist weniger Umfangreich, wodurch eine Implementierung in C/C++ für diese Bachelorarbeit zu aufwändig gewesen wäre. Eine Implementierung in C/C++ könnte die Performance des Systems verbessern, da die Implementierung in C/C++ den Vorteil hätte, dass die verschiedenen genutzten Solver direkt eingebunden und über eine Schnittstelle aufgerufen werden können. So kann z.B.: der C++ Quellcode von MiniSat online frei abgerufen werden [8]. Da beim Aufrufen der subsume Methode ein großer Teil der Laufzeit für das bilden der Formel verbraucht wird, könnte dies eine relevante Optimierung der Laufzeit des Systems bringen.

Wenn die angenommenen Performance-Vorteile groß genug sind, könnte sich eine Implementierung in C/C++.

[8] MiniSat - Quellcode: https://github.com/niklasso/minisat

4.5 Bedienung

Das implementierte System liegt dieser Arbeit als eine .jar bei. In diesen Abschnitt wollen wir kurz beschreiben, wie diese aufgerufen werden kann.

Subsumtionen entscheiden

Argumente: --subsume (kurz -s) <PfadZurOWL><DasSubsumierte><DasSubsumierende>

Beschreibung: Bei diesen Aufruf muss den Programm ein Pfad zu einer vorliegenden .owl Ontologie und zwei Konzeptnamen übergeben werden, und entscheidet dann, ob in der vorliegenden Ontologie $ClassA \sqsubseteq ClassB$. Dabei wird ein Fehler geworfen, wenn der gegebene Pfad nicht zu einer gültigen Ontologie mit einer entsprechenden Ausdruckstärke führt, oder wenn $ClassA$ oder $ClassB$ kein Konzept aus der Ontologie beschreibt.

Beispiel: Im folgenden ein Beispiel wie das Programm mit diesen Argumenten aufgerufen werden soll:

```
> java -jar funky-all-0.1.jar -s horrocks-galen-modified2.owl InguinalTri
```

Die Ausgabe sieht in diesem Fall wie folgt aus:

```
InguinalTriangle
does subsume
AnatomicalSurfaceTriangle
(The decision took 5ms)
```

Testmodus

Argumente: --test (kurz -t)

Beschreibung: Dieser Aufruf startet einfach nur die JUnit Tests und gibt aus, ob diese erfolgreich waren. Dies soll es ermöglichen die .jar auf ihre Funktionsfähigkeit zu testen.

Beispiel: Ein Aufruf sieht Beispielhaft wie folgt aus:

```
> java -jar funky-all-0.1.jar -t
```

Die Ausgabe auf diesen Aufruf sollte dise Form haben:

```
Detected that os is a linux derivate
Test over all subsumtions for the small ontology took 23
Fact++ did need aprox. 33ms for this (with optimizations)
All test were succesful
```

Evaluationsmodus

Argumente: --evaluate (kurz -e) <PfadZurOWL>

Beschreibung: Dieser Aufruf führt eine Evaluation über eine gegebene Ontologie ähnlich dem im Kapitel 5 beschrieben durch und gibt das Ergebnis der Laufzeit im Vergleich

zu Fact++ und CB zurück. Dabei werden je 10 Konzepte mit einer niedrigen, mittleren und hohen Rollentiefe für das Experiment genutzt. Dieser Aufruf gibt eine .tex aus, die das Ergebnis als Graphen und Tabellen darstellen. Es wird ein Fehler ausgegeben, wenn kein korrekter Pfad zu einer Ontologie übergeben wird. Wenn kein Pfad zu einer OWL übergeben wird, wird die im Kapitel 5.2 beschriebene modifizierte Version der Ontologie Horrocks-Galen genutzt.

Anzumerken ist, dass die Evaluation die Solver Fact++ und CB nutzt. Daher wurden diese beiden Solver in das vorliegende Programm mit eingebunden. Für eine weitere Veröffentlichung wäre es möglich (und sinnvoll) diese Einbindung zu entfernen.

Beispiel: Der Aufruf sollte die folgende Form haben (dabei liegt die OWL-Datei im selben Ordner in den der Aufruf getätigt wird):

```
> java -jar funky-all-0.1.jar -e horrocks-galen-modified2.owl
```

Die Ausgabe ist hier relativ groß, da z.B CB der Fortschritt von CB ausgegeben wird. Die endgültige Ausgabe sind der Vergleich der Laufzeiten, die auch in einer Datei logging.tex vorliegt die man mit einer entsprechenden Latex Distribution übersetzen kann.

5 Tests und Evaluation

In diesen Kapitel soll das implementierte System Funky getestet und evaluiert werden. Im ersten Schritt testen wir das System anhand kleinerer Ontologien. Im zweiten Schritt evaluieren wir das System und vergleichen es mit bereits implementierten OWL-Reasonern. Zum Schluß diskutieren wir diese Ergebnisse.

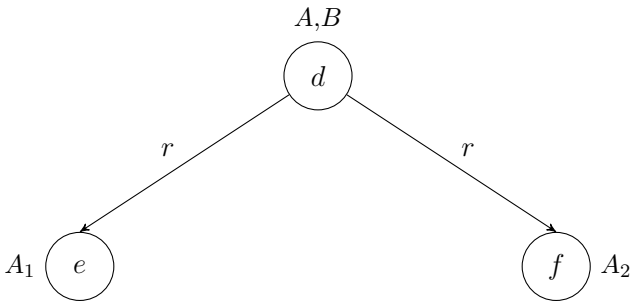
5.1 Tests

Das System wurde anhand einer Klassifikation (also wir testen für alle $A, B \in N_C$: $A \sqsubseteq B$) von 2 Ontologien getestet. Zunächst wurde die Beispiel Ontologie/TBox aus Kapitel 3 genutzt, um das System zu testen. Die Ontologie beinhaltet 7 Konzepte und 2 Klassen mit 5 Axiome. Um festzustellen ob sich 2 Klassen subsumieren haben wir die Ontologie zunächst mit Fact++ klassifiziert und das Ergebnis genutzt um ein JUnit Testfall aufzubauen, in der wir die Ontologie sogar vollständig klassifizieren, indem die Subsumtion zwischen allen Konzepten getestet wird.

Im nächsten Test bauen wir eine neue TBox $\mathcal{T}_2$ auf, in der es für die Klassifikation einen Unterschied macht, ob die Rollen funktional sind, oder nicht. Diese TBox $\mathcal{T}_2$ sieht wie folgt aus:

$$\begin{aligned} A &\equiv \exists r.A_1 \sqcap \exists r.A_2 \\ B &\equiv \exists r.(A_1 \sqcap A_2) \end{aligned} \tag{19}$$

Offenbar gilt für diese TBox $\mathcal{T}_2 \models A \sqsubseteq B$ wenn die Rolle r funktional ist. Dies gilt aber nicht wenn r nicht funktional ist. Ein Modell $\mathcal{I}$ für das $\mathcal{I} \not\models \mathcal{T}_2$ nur gilt, wenn r funktional ist sieht wie folgt aus:



(mit $\Delta^{\mathcal{I}} = \{d, e, f\}$, $A^{\mathcal{I}} = \{d\}$, $(A_1)^{\mathcal{I}} = \{e\}$, $(A_2)^{\mathcal{I}} = \{f\}$, $r^{\mathcal{I}} = \{(d, e), (d, f)\}$)

Offenbar gilt für diese Interpretation $\mathcal{I} \models \mathcal{T}_2$, aber $\mathcal{I} \not\models A \sqsubseteq B$. Wenn r funktional ist, kann es nur eine "r-Kante" von d geben, sodass das entstehende Modell auch ein Modell von $A \sqsubseteq B$ ist.

Auch hier sind wir ähnlich vorgegangen wie im ersten Test. Für diesen Testfall zeigt sich, dass unser System Funky beim Entscheiden über Subsumtionen die Funktionalität der Rollen einhält, und die TBox somit korrekt klassifiziert wird.

Ein weiterer größerer Test findet während der Evaluation im nächsten Kapitel statt. Dort werden wir die Laufzeit von Funky mit zwei bereits existierenden Reasonern vergleichen. Es werden fünf mal zehn Konzeptpaare verwendet. Neben der Laufzeit vergleichen wir zudem die Ergebnisse der Reasoner. Es zeigt sich, dass Funky und die beiden existierenden Reasoner immer die selbe Antwort zurückgeben.

5.2 Evaluation

Nun wollen wir die Laufzeit von Funky evaluieren und mit der Laufzeit von vorhandenen OWL-Reasonern vergleichen. Genutzt werden sollen die Reasoner Fact++ zuerst vorgestellt von Tsarkov et al. (2007) und CB vorgestellt von Kazakov (2009). Fact++ wurde gewählt, weil Fact++ ein bekannter und verbreiteter Tableau-basierter Reasoner ist. CB ist ein Reasoner der die Ontologie Galen in kurzer Zeit klassifizieren kann (zum Vergleich: Tableau-basierte Reasoner können diese nicht klassifizieren), wobei $\mathcal{EL}^{\mathcal{F}}$ eine Teilmenge der Ausdrucksstärke von Galen besitzt.

Das Experiment findet auf einem System mit den folgenden technischen Daten statt: Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz mit 2 Kernen und 16GiB Arbeitspeicher (SODIMM DDR3 1600 MHz).

Für das Experiment werden wir die Horrocks-Galen Ontologie[9] nutzen. Diese ist eine verkleinerte Version der Galen Ontologie, mit 7690 Axiomen, 2749 Konzepten, und 413 Rollen (zum Vergleich hat das volle Galen 63329 Axiome, 23141 Konzepte, 950 Rollen [10]), das eine Ontologie von medizinischen Begriffen, Anatomie und Medikamenten ist. Wir nutzen diese, da sie einer "reelen" Ontologie entspricht, an der wenige Änderungen vorgenommen werden müssen, damit sie der Ausdruckstärke von $\mathcal{EL}^{\mathcal{F}}$ entspricht. Zudem ist die größe (und vorkommende Konzepttifen) von Horrocks-Galen für unser System und Fact++ noch handhabbar.

Um diese Ontologie für unsere Evaluation nutzen zu können, müssen wir diese so modifizieren, dass sie der Ausdrucksstärke von $\mathcal{EL}^{\mathcal{F}}$ entspricht. Dazu wurden zunächst alle Subsumtionsbeziehungen zwischen Rollen (SubObjectPropertyOfAxiom), transitiven Rollen (TransitiveObjectPropertyAxioms) und (vorerst) die Funktionalität der Rollen (FunctionalObjectPropertyAxiom) aus der Ontologie entfernt. Zudem werden alle GCI's (General Concept Inclusions, Axiome bei der auf der Linken Seite auch zusammengesetzte Konzepte vorkommen können) entfernt. Um die Ontologie jetzt in eine azyklische Ontologie zu übersetzen, sorgen wir zunächst dafür, dass Konzepte nur einmal auf der linken Seite vorkommen, indem wir, wenn zwei Axiome der Form $A \sqsubseteq B_1$ und $A \sqsubseteq B_2$ in der Ontologie vorkommen eine zufällige davon löschen. Auch muss jede Zyklität entfernt werden, indem wir für jedes Konzept den Abhängigkeitsgraph durchgehen, und wenn wir das selbige Konzept wieder erreichen, das letzte "durchschrittene" Axiom entfernen. Nun haben wir aus Horrocks-Galen eine azyklische Ontologie gebildet, die wenn man alle Rollen funktional macht der Ausdrucksstärke von $\mathcal{EL}^{\mathcal{F}}$ entspricht.

Die so gebildete Ontologie (mit allen Rollen funktional) hat eine durchschnitliche Konzepttiefe von ca. 5,25 und eine durchschnittliche Rollentiefe von ca. 0,47. Für das Experiment haben wir 3 Mengen mit je 20 Konzepten aus der Ontologie entnommen. Die erste Menge entspricht den 20 Konzepte mit der niedrigsten Konzepttiefe, in diesem Fall eine

[9]Horrocks Galen: http://www.cs.man.ac.uk/ horrocks/OWL/Ontologies/galen.owl
[10]Full Galen zu funden: http://owl.cs.manchester.ac.uk/research/co-ode/

durchschnittliche Konzepttiefe von 2 und eine durchschnittliche Rollentiefe von 0,25. Die zweite Menge ist eine Menge von 20 Konzepte mit ungefähr durchschnitlicher Konzepttiefe (durchschnittliche Konzepttiefe von 7 und durchschnittliche Rollentiefe von 0,2). Die letzte Menge sind die 20 Konzepte mit der höchsten durchschnittliche Konzepttiefe (durchschnittliche Konzepttiefe ist 18,75, durchschnittliche Rollentiefe ist 4,6).

In der Vorbereitung bilden wir für jede Menge 10 (disjunkte) Paare, sodass wir diese für den Vergleich der verschiedenen Reasoner nutzen können. Zudem bilden wir zwei weitere Menge von Konzeptpaaren. In der ersten Menge bilden wir die Paare aus je einem Konzept aus der Menge der Konzepte mit minimaler Konzepttiefe und einem Konzept aus der Menge der Konzepte mit maximaler Konzepttiefe. Hiermit wollen wir die Laufzeit der Entscheidung über die Subsumtion zwischen Konzepten mit *minimaler* Konzepttiefe und Konzepten mit *maximaler* Konzepttiefe betrachten. Die zweite Menge wird ähnlich gebildet, aber hier soll die Laufzeit der Entscheidung über die Subsumtion zwischen Konzepten mit *maximaler* Konzepttiefe und Konzepten mit *minimaler* Konzepttiefe betrachten.

Für das Experiment sammeln wir die Laufzeiten für alle diese Konzeptpaare und betrachten zudem die durchschnittliche Laufzeit. Anzumerken ist, dass wir für Fact++ den Reasoner jeweils neu instanziieren, da Fact++ sonst Ergebnisse vorheriger Anfragen nutzt um die Subsumtionen zu entscheiden, und so einen Geschwindigkeitsvorteil erhalten würde. Diesen Vorteil wollen wir in unsere Evaluation nicht in Betracht ziehen, da wir solche Optimierungen nicht in unseren Reasoner Funky implementiert haben, und wir nur die Algorithmen vergleichen wollen. Ähnlich gehen wir beim CB Reasoner vor. CB klassifiziert aber bei jedem Durchlauf die ganze Ontologie, daher ist dieser Reasoner schwer mit den anderen beiden Reasonern zu vergleichen. Es zeigt sich aber, dass CB zur Klassifikation der gesamten Ontologie weniger Laufzeit benötigt, als Funky für ein Konzeptpaar mit jeweils hoher Konzepttiefe. Daher werden wir CB in den folgenden Punkten zunächst nicht genau diskutieren. Nun wollen wir die Ergebnisse der 5 verschiedenen Mengen von Konzeptpaaren betrachten.

Konzeptpaare mit niedriger Konzepttiefe

Tabelle 3: Performanzvergleich - niedrige Konzepttiefe

Beispiel	Funky	Fact++	CB
0	2ms	30ms	252ms
1	0ms	22ms	191ms
2	0ms	27ms	220ms
3	0ms	37ms	201ms
4	1ms	28ms	235ms
5	1ms	34ms	199ms
6	0ms	21ms	345ms
7	0ms	31ms	338ms
8	1ms	26ms	215ms
9	1ms	36ms	300ms
Durchschnitt (Ø)	0.6	29.2	249.6
Standardabweichung (STDEV.)	0.66	5.19	54.87

Für diese Konzeptpaare gilt das die durchschnittliche Konzepttiefe 2 und die durch-

schnittliche Rollentief 0,25 ist. Es zeigt sich, dass Funky hier einen Vorteil gegenüber den Tableau-basierten Reasoner Fact++ hat.

Konzeptpaare mit mittlerer Konzepttiefe

Tabelle 4: Performanzvergleich - mittlere Konzepttiefe

Beispiel	**Funky**	**Fact++**	**CB**
0	4ms	34ms	200ms
1	5ms	20ms	150ms
2	4ms	21ms	268ms
3	6ms	36ms	259ms
4	4ms	24ms	240ms
5	12ms	29ms	177ms
6	6ms	27ms	156ms
7	4ms	15ms	138ms
8	6ms	27ms	182ms
9	4ms	23ms	182ms
Ø	5.5	25.6	195.2
STDEV.	2.33	6.07	43.51

Für diese Konzeptpaare gilt das die durchschnittliche Konzepttiefe 7 und die durchschnittliche Rollentief 0,2 ist. Es zeigt sich, dass Funky auch hier einen Vorteil gegenüber den Tableau-basierten Reasoner Fact++ hat.

Konzeptpaare mit hoher Konzepttiefe

Tabelle 5: Performanzvergleich - hohe Konzepttiefe

Beispiel	**Funky**	**Fact++**	**CB**
0	226ms	16ms	173ms
1	1083ms	19ms	143ms
2	485ms	12ms	129ms
3	620ms	14ms	133ms
4	546ms	19ms	164ms
5	1088ms	21ms	131ms
6	224ms	16ms	192ms
7	945ms	25ms	116ms
8	990ms	19ms	154ms
9	925ms	17ms	192ms
Ø	713.2	17.8	152.7
STDEV.	319.04	3.49	25.48

Für diese Konzeptpaare gilt das die durchschnittliche Konzepttiefe 18,75 und die durchschnittliche Rollentief 4,6 ist. In diesem Fall zeigt sich aber, dass Funky wesentlich schlechter abschneidet als Fact++.

Konzeptpaare mit je einer niedrigen und einer hohen Konzepttiefe

Tabelle 6: Performanzvergleich - niedrige/hohe Konzepttiefe

Beispiel	**Funky**	**Fact++**	**CB**
0	568ms	19ms	137ms
1	203ms	17ms	138ms
2	519ms	16ms	137ms
3	939ms	17ms	145ms
4	277ms	13ms	140ms
5	483ms	17ms	124ms
6	979ms	17ms	114ms
7	538ms	15ms	123ms
8	568ms	19ms	113ms
9	477ms	17ms	115ms
Ø	555.1	16.7	128.6
STDEV.	232.73	1.68	11.5

In diesem Fall zeigt sich, dass Funky wesentlich schlechter abschneidet als Fact++.

Konzeptpaare mit je einer hohen und einer niedrigen Konzepttiefe

Tabelle 7: Performanz - hohe/niedrige Konzepttiefe

Beispiel	**Funky**	**Fact++**	**CB**
0	1ms	19ms	173ms
1	0ms	25ms	181ms
2	2ms	18ms	202ms
3	1ms	16ms	176ms
4	1ms	22ms	149ms
5	0ms	24ms	140ms
6	2ms	17ms	146ms
7	0ms	23ms	127ms
8	2ms	26ms	130ms
9	0ms	18ms	121ms
Ø	0.9	20.8	154.5
STDEV.	0.83	3.43	25.6

In diesem Fall hat Funky wieder einen großen Vorteil gegenüber Fact++.

Auswertung

Das Experiment hat uns gezeigt, dass die durchschnittliche Laufzeit von Funky bei niedrigen Konzepttiefen niedriger als die vom Tableau-basierten Reasoner Fact++ ist. Dies ändert sich aber, sobald die Konzepte auf der rechten Seite der Subsumtion ein große Konzepttiefe besitzen.

Im Vergleich zu Fact++ sieht man zudem, dass die Standardabweichung (im Verhältnis zur durchschnittlichen Laufzeit) bei Funky größer ist. Dies könnte daran liegen, dass die SAT-Solver eine stärkere Varianz bei Lösen der Formeln haben.

Um die Laufzeiten von Funky genauer zu analysieren, wurde die Laufzeit mit den Profiler JProfiler betrachtet. Dabei zeigte sich, dass Funky bei hohen Konzepttiefen auf der rechten Seite etwas mehr als ein Drittel seiner Laufzeit für das hinzufügen der Literale zu den SAT-Solver von Kodkod benötigt. Dies ist insbesondere bei den Teilformeln von $repres^{\mathcal{T}}_{B_0}$ relevant, da diese eine größere Verschachtelung bei den logischen Quantoren anhand der Konzepttiefe des Konzepts auf der rechten Seite besitzt.

Insgesamt zeigt sich aber auch, das CB einen Vorteil gegenüber Funky, da CB zur Klassifikation der gesamten Ontologie meist um die 150ms benötigt, während Funky für die Entscheidung einer einzigen Subsumtion für ein Konzept mit hoher Konzepttiefe auf der linken Seite im Schnitt 713ms benötigt. Auf der vorliegenden Ontologie ist die Laufzeit der gesamten Klassifikation für Fact++ auf einer ähnlichen höhe wie CB.

6 Zusammenfassung und Ausblick

Abschließend wollen wir das Ergebnis der Arbeit diskutieren, und einen Ausblick auf die mögliche Weiterentwicklung des Reasoner Funky geben.

6.1 Zusammenfassung

Die Leitfrage dieser Arbeit war, ob es möglich ist, für eine nicht-effiziente Erweiterung von leichtgewichtigen Beschreibungslogiken spezialisierte Reasoner zu entwickeln, die den "schweren" Teil an einen SAT-Solver auslagern und sich dank der Optimierungen im SAT-Solver relativ performant verhalten.

In den Kapiteln zu den Theoretischen Grundlagen unseres Systems haben wir gezeigt, dass die Implementierung eines solchen Reasoner möglich ist. Zudem zeigen die Tests in Kapitel 5.1, dass sich die Implementierung korrekt verhält. Auch verhält sich Funky durch die verschiedenen Optimierungen im Kapitel 4.3 relativ performant und kann zumindest einzelne Subsumtionsanfragen in angemessener Zeit beantworten.

Im Kapitel 5.2 haben wir das System mit bereits vorhandenen Reasoneren verglichen und dabei gelernt, dass sich Funky zumindest bei niedrigeren Konzepttiefen schneller verhält als Tableau-basierte Reasoner. Leider erhöhte sich die Laufzeit sehr stark, sobald die Konzepttiefe des Konzepts auf der rechten Seite groß wurde. Weitere Analysen zeigten, dass ein großer Teil dieser Laufzeit für das Hinzufügen der Literale zum Solver verbraucht wurde. Daher könnte sich hier, wie bereits in Kapitel 4.4 diskutiert, ein Wechsel zu einer anderen Programmiersprache lohnen, die es möglich macht den Solver ohne Umwege über die Kodkod-API anzusprechen.

Es hat sich aber auch gezeigt das die Laufzeiten einzelner Subsumtionen von Funky im schlimmsten Fall wesentlich höher als die Laufzeit einer Klassifikation mit dem konsequenzbasierten Reasoner CB ist. Dies bedeutet, dass das momentane System nur zur Klassifikation genutzt werden kann, wenn diese Subsumtionstests vermieden werden können. Hier sind weitere Betrachtungen notwendig, die unter anderem im nächsten Kapitel skizziert werden.

6.2 Ausblick

Zunächst müssen weitere Optimierungen stattfinden, sodass sich das System auch bei hohen Konzepttiefen performant verhält. Eine Möglichkeit wäre die alternative Implementierung in C/C++, wie es in Kapitel 4.4 angesprochen wurde. Zudem könnte untersucht werden, ob noch weitere Optimierungen im Aufbau der Formel getätigt werden können.

Falls sich diese Probleme beheben lassen, könnte das hier entwickelte System als Grundlage für einen OWL-Reasoner dienen. Dazu müsste zuallererst das System so umgebaut werden, dass nicht nur einzelne Subsumtionen entschieden werden, sondern Ontologien klassifizieret werden können.

Zudem müsste das System an die Schnitstelle von Reasoner in der OWL API angepasst werden [11], um so z.B. in Protegé eingebunden werden zu können.

[11] OWLReasoner: http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/reasoner/OWLReasoner.html

In ihren Artikel haben Haase and Lutz (2008, vgl. S.28) einen Algorithmus vorgestellt, der einen Algorithmus, der die Subsumtion für $\mathcal{EL}^{\mathcal{F}}$ entscheidet als Subprozedur nutzt, um die Subsumtion für $\mathcal{EL}^{f}$ entscheiden. Dabei ist $\mathcal{EL}^{f}$ die Logik, die $\mathcal{EL}$ um die Möglichkeit erweitert, einige Rollen funktional zu setzen (und nicht alle wie bei $\mathcal{EL}^{\mathcal{F}}$). Für den Algorithmus ist es aber notwendig nicht nur das Subsumtionsproblem entscheiden zu können, sondern eine Ontologie in $\mathcal{EL}^{\mathcal{F}}$ klassifizieren zu können.

Für die Klassifikation können verschiedene Optimierungen genutzt werden, die auch bei anderen Reasonern Verwendung finden. Ein Beispiel dafür sind die bei Fact++ verwendeten "told subsumer" (Tsarkov et al., 2007, vgl. S.17). Diese stellen "offensichtliche" Subsumtionen in einer Ontologie dar, sodass man diese bereits vor dem Start des eigentlichen Algorithmus. Beispielsweise ist D ein "told subsumer"für C, wenn das Axiom $C \equiv D \sqcap C$ in der TBox vorkommt. Tsarkov et al. (2007, vgl. S.31-39) diskutieren in ihrem Artikel zudem einige weitere Optimierungen für die Klassifikation. Eine dieser Optimierungen wäre z.B., dass $C \sqsubseteq D$ nur getestet wird, wenn 6 gezeigt wurde, dass D von allen Konzepten aus N_C, die C subsumieren, subsumiert wird. Zudem werden Optimierungen für die Reihenfolge der Konzepte bei der Klassifikation beschrieben. Insbesondere bei der Reihenfolge könnte ein hohes Optimierungspotenzial vorliegen, da sich gezeigt hat, dass unser Reasoner die Subsumtion von Konzepten mit niedriger Konzepttiefe schneller entscheiden kann.

6.3 Fazit

In dieser Arbeit haben wir gezeigt, dass es möglich ist einen Reasoner für eine leichtgewichtige Beschreibungslogik mit Funktionalität zu konstruieren und zu implementieren. Es hat sich gezeigt, dass der implementierte Reasoner Funky für Subsumtionstests mit Konzepten, die eine relativ niedrige Konzepttiefe besitzen, auf der linken Seite schneller als der Tableau-basierte Reasoner Fact++ ist. Es hat sich aber auch die Erwartung bestätigt, dass Funky für den Subsumtionstest mit einem Konzept mit hoher Konzepttiefe auf der linken Seite mehr Laufzeit benötigt, als der Konsequenz-basierte Reasoner CB für die gesamte Klassifikation der Ontologie.

Trotzdem ist Funky interessant für den begrenzten Anwendungsfall des Tests von einzelnen Subsumtionen für Konzepte, die in ihrer Ontologie eine relativ geringe Konzepttiefe besitzt. Es besteht zudem die Hoffnung, dass die Klassifikation einer Ontologie möglich ist, wenn eine Heuristik gefunden werden kann, die versucht Konzepte mit hoher Konzepttiefe auf der linken Seite der Subsumtionsanfrage zu vermeiden. Hierfür könnten weitere Untersuchungen sinnvoll sein.

Inhalt der CD

Die CD hat folgenden Inhalt:

- Die schriftliche Bachelorarbeit als PDF
- Der implementierte Reasoner als .jar
- Das entsprechende Eclipse-Projekt

Literatur

Audemard, G. and Simon, L. (2009). Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 399–404, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Baader, F. (2003). *The Description Logic Handbook : Theory, Implementation and Applications.* Cambridge University Press.

Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the el envelope. LTCS-Report LTCS-05-01, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany. See http://lat.inf.tu-dresden.de/research/reports.html.

Baader, F., Horrocks, I., and Sattler, U. (2008). Description Logics. In van Harmelen, F., Lifschitz, V., and Porter, B., editors, *Handbook of Knowledge Representation*, chapter 3, pages 135–180. Elsevier.

Biere, A. (2010). Lingeling, plingeling, picosat and precosat at sat race 2010. Technical report.

Brachman, R. J. and Schmolze, J. G. (1985). An overview of the kl-one knowledge representation system. *Cognitive Science*, 9(2):171–216.

Brandt, S. (2004). Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In de Mantáras, R. L. and Saitta, L., editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, pages 298–302. IOS Press.

Cimino, J. and Zhu, X. (2006). The practical impact of ontologies on biomedical informatics. *IMIA Yearbook of Medical Informatics*, 1(1):124–135.

Eén, N. and Sörensson, N. (2003). An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518.

Haase, C. (2007). Complexity of subsumption in extensions of $\mathcal{EL}$. Master's thesis (Diplomarbeit), Technische Universität Dresden, Germany.

Haase, C. and Lutz, C. (2008). Complexity of subsumption in the $\mathcal{EL}$ family of description logics: Acyclic and cyclic tboxes. In *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, pages 25–29.

Hayes, P. J. (1979). The logic of frames. In Metzing, D., editor, *Frame Conceptions and Text Understanding*, pages 46–61. Walter de Gruyter and Co., Berlin, Germany.

Horridge, M. and Bechhofer, S. (2011). The owl api: A java api for owl ontologies. *Semant. web*, 2(1):11–21.

Kazakov, Y. (2009). Consequence-driven reasoning for horn SHIQ ontologies. In *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009.*

Le Berre, D. and Parrain, A. (2010). The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64.

Levesque, H. J. and Brachman, R. J. (1985). A fundamental tradeoff in knowledge representation and reasoning (revised version). In Brachman, R. J. and Levesque, H. J., editors, *Readings in Knowledge Representation*, pages 41–70. Kaufmann, Los Altos, CA.

Lutz, C. (1999). Complexity of terminological reasoning revisited. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning LPAR'99*, Lecture Notes in Artificial Intelligence, pages 181–200. Springer-Verlag.

Mendez, J. A. (2007). A classification algorithm for $\mathcal{ELHI}f_{\nabla^+}$. Master's thesis (Diplomarbeit), Technische Universität Dresden, Germany.

Minsky, M. (1981). A framework for representing knowledge. In Haugeland, J., editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 95–128. MIT Press, Cambridge, MA.

Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., and Lutz, C. (2012). Owl 2 web ontology language: Profiles. W3c working draft, W3C.

Nebel, B. (1990). Terminological Reasoning Is Inherently Intractable. *Artificial Intelligence*, 43(2):235–249.

Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation. Available at `http://www.w3.org/TR/owl-semantics/`.

Plaisted, D. A. and Greenbaum, S. (1986). A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304.

Quillian, M. R. (1967). Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities. *Behavioral Science*, 12:410–430.

Schmidt-Schauß, M. and Smolka, G. (1991). Attributive concept descriptions with complements,. *Artificial Intelligence*, 48:1–26.

Spackman, K. A., Campbell, K. E., and Côté, R. A. (1997). SNOMED RT: A reference terminology for health care. In *J. of the American Medical Informatics Association*, pages 640–644.

Suntisrivaraporn, B. (2009). *Polynomial time reasoning support for design and maintenance of large-scale biomedical ontologies*. PhD thesis, Dresden University of Technology.

Torlak, E. and Jackson, D. (2007). Kodkod: A relational model finder. In *In Tools and Algorithms for Construction and Analysis of Systems (TACAS*, pages 632–647. Wiley.

Tsarkov, D., Horrocks, I., and Patel-Schneider, P. F. (2007). Optimizing terminological reasoning for expressive description logics. *J. Autom. Reasoning*, 39(3):277–316.

Abbildungsverzeichnis

Tabellenverzeichnis