Diplomarbeit

Subsumtion in $\mathcal{EL}$ bezüglich hybrider TBoxen

Jörg Model

2005

TU Dresden
Lehrstuhl für Automatentheorie
Fakultät Informatik

Betreuender Hochschullehrer: Prof. Dr.-Ing. Franz Baader

TECHNISCHE UNIVERSITÄT DRESDEN

Name: **Jörg Model**

Matrikelnummer: **2634548**

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe. Sämtliche Zitate wurden als solche kenntlich gemacht. Mathematische Definitionen, die aus anderen Quellen stammen, wurden nicht einzeln als solche kenntlich gemacht; jedoch ist zu Beginn des Kapitels 2 ein Vermerk dazu gemacht. In der Software auf der beigefügten CD wird fremder Code wiederverwendet — die Hauptdatei ist jedoch ausschließlich von mir geschrieben, die anderen Dateien enthalten im Dateikopf den Namen des Authors.

Hofstetten, 14. April 2005

Ort und Datum *Unterschrift*

Dank

An dieser Stelle möchte einigen Menschen meinen Dank aussprechen, die auf die eine oder andere Weise dazu beigetragen haben, dass ich diese Arbeit jetzt vorlegen kann. An erster Stelle gilt mein Dank meinem Vater, der mich oft genug während meines Studiums unterstützt hat. Herrn Prof. Franz Baader für das Vertrauen, dass ich dieser Arbeit gewachsen bin, sowie für seine richtungsweisenden Hinweise, wenn die Entwicklung stecken geblieben war. Sebastian Brandt für die langen und ausführlichen Gespräche, die ebenfalls wesentlich zu meinem Verständnis der Materie beigetragen haben und Boontawee Suntisrivaraporn für die Zusammenarbeit bei den Implementierungen. Bedanken möchte ich außerdem bei Frau Achtruth für's Unsichtbare, sowie bei Phiniki Stouppa, die mir in gewisser Weise bei der Literaturauswahl geholfen hat und bei Jenny Müller für Ihre Gastfreundschaft.

Abstrakt

Für die Beschreibungslogik $\mathcal{EL}$ ist gezeigt worden, dass das Subsumtionsproblem für zyklische TBoxen in polynomieller Laufzeit entscheidbar ist, sowohl mit *gfp*-Semantik, als auch mit deskriptiver Semantik. Auch das *kleinste gemeinsame Oberkonzept* (lcs) von zwei Konzepten aus einer zyklischen $\mathcal{EL}$-TBox existiert immer und kann in polynomieller Zeit berechnet werden, wenn man die *gfp*-Semantik zugrunde legt. Ebenso wurde auch für generelle $\mathcal{EL}$-TBoxen mit GCIs bezüglich deskriptiver Semantik ein polynomieller Subsumtionsalgorithmus angegeben. Da es beim Vorkommen von GCIs nicht sinnvoll ist, TBoxen mit *gfp*-Semantik zu interpretieren, andererseits nur die *gfp*-Semantik für zyklische $\mathcal{EL}$-TBoxen die wünschenswerte Eigenschaft garantiert, dass das lcs immer existiert und in polynomieller Zeit berechnet werden kann, wurde eine eine neue Art von TBoxen konzipiert, in der sowohl zyklische Definitionen vorkommen, die mit *gfp*-Semantik interpretiert werden, als auch GCIs, die mit deskriptiver Semantik interpretiert werden. Wir zeigen, dass das Subsumtionsproblem für diese hybriden $\mathcal{EL}$-TBoxen in polynomieller Zeit auf das Subsumtionsproblem für zyklische TBoxen mit *gfp*-Semantik reduziert werden kann und geben ein Verfahren dafür an. Außerdem wird eine prototypische Implementierung in LISP vorgestellt, die auf bestehenden Implementierungen für das Subsumtionsproblem von generellen TBoxen und zyklischen $\mathcal{EL}$-TBoxen mit *gfp*-Semantik aufbaut.

Inhaltsverzeichnis

„Doch welcher Stein ist es, der die Brücke trägt?“, fragt Kublai Khan.

„Die Brücke wird nicht von diesem oder jenem Stein getragen“, antwortet Marco, „sondern von der Linie des Bogens, den diese bilden.“ [10]

Kapitel 1

Einleitung

Ein wichtiges Teilgebiet in der Forschung zur künstlichen Intelligenz (KI) war und ist die Wissensrepräsentation (KR engl.: *knowledge representation*). KR bedeutet, Wissen aus bestimmten Domänen in einer Form darzustellen, so dass automatisiertes Schlussfolgern über dem Wissen möglich ist. Beschreibungslogiken (DL engl.: *description logic*) sind formale Sprachen zur Repräsentation von Wissen, d.h. zur Beschreibung von Begriffen eines bestimmten Diskursbereichs und den strukturellen Abhängigkeiten der Konzepte voneinander in einer formalen Sprache. Beschreibungslogiken sind eine Weiterentwicklung von *semantischen Netzwerken*, die in der zweiten Hälfte des 20. Jahrhunderts[1] zur Wissensrepräsentation entwickelt wurden [21]. Im Unterschied zu den semantischen Netzwerken sind DLs jedoch mit einer formalen Semantik versehen, was wiederum für den sinnvollen Einsatz von Algorithmen zum automatischen Schließen über Wissensbasen unbedingt notwendig war. Für eine generelle Einführung in das Thema ‚Beschreibungslogik‘ siehe z.B. [4, 5]. Begriffe werden in einer DL durch *Konzepte* (unäre Prädikate) beschrieben und ihre strukturellen Abhängigkeiten untereinander durch *Rollen* (binäre Relationen); eine DL wird hauptsächlich durch die *Konstruktoren* charakterisiert, die in ihr zur Verfügung stehen, um aus einfachen Konzeptbeschreibungen komplexere zu bilden. Diese

[1] Eigentlich werden graphische Notationssysteme zur Repräsentation von Wissen schon sehr viel länger benutzt — schon aus der Antike ist das Beispiel des *Baum des Porphyrios* von Porphyrios von Tyros bekannt — seit etwa den sechziger Jahren des letzten Jahrhunderts werden diese Notationssysteme aber auch in Computerprogrammen implementiert.

Konstruktoren bestimmen in der Regel sowohl die Ausdrucksstärke der DL als auch ihre Komplexität. Die Beschreibungslogik $\mathcal{EL}$ stellt neben dem '$\top$' Konzept noch die Konjunktion ('$\sqcap$') und die Existenzrestriktion ('$\exists$.') als Konstruktoren zur Verfügung. Das $\mathcal{EL}$-Konzept

$$\mathsf{Bird} \sqcap \exists \mathsf{hunts.}(\mathsf{Fish} \sqcap \mathsf{Frog}) \sqcap \exists \mathsf{atEnviroment.Waters}$$

beschreibt z.B. die Klasse der Vögel, die Fische und Frösche jagen und an Gewässern leben.

Eine Kollektion von Konzeptbeschreibungen in einer DL-Sprache aus einer Wissensdomäne heißt Terminologie oder TBox. Durch eine TBox wird eine Wissensbasis gebildet, auf denen DL-Systeme arbeiten, die in den entsprechenden Gebieten eingesetzt werden[2]. Im *Gene Ontology* Projekt [11] z.B. erstellen Wissenschaftler aus verschiedenen Forschungseinrichtungen aus dem Bereich der Biologie in einer gemeinschaftlichen Anstrengung eine einheitliche Beschreibungsgrundlage für automatisierte Anfragen in biologischen Datenbanken. Die dieser Wissensbasis zugrunde liegende DL ist die Sprache $\mathcal{EL}$. Ein anderes Beispiel für eine Wissensbasis findet sich im SNOMED-Projekt (*Systematized Nomenclature of Medicine*, siehe [12]); auch hier ist die zugrunde liegende Logiksprache $\mathcal{EL}$. DL-Systeme sind Anwendungen, die auf Basis einer DL-Sprache zusammen mit einer Wissensbasis *Inferenzverfahren* zur Verfügung stellen, um aus den Konzepten der Wissensbasen zu Schlussfolgerungen zu ziehen und so implizites Wissen explizit zu machen. Eines der wichtigsten Inferenzprobleme in DLs ist die Subsumtion zwischen Konzepten einer TBox, bei dem man die Oberkonzept-Unterkonzept-Beziehung zwischen den Konzepten feststellen will, um z.B. eine Taxonomie der TBox zu erstellen.

TBoxen lassen sich nach Art der Beschaffenheit der in ihnen enthaltenen Konzeptbeschreibungen unterscheiden: einfache TBoxen enthalten nur Konzeptdefinitionen. In einer Konzeptdefinition wird einem Konzeptnamen eine definierende Beschreibung zugeordnet. Wenn die Konzeptdefinitionen einer TBox terminologische Zyklen enthalten, spricht man von zyklischen TBoxen. Ein terminologischer Zyklus entsteht, wenn ein Konzept direkt oder indirekt durch sich

[2] Genauer gesagt, lassen sich solche Wissensbasen in TBoxen übersetzen, da die realen Wissensbasen üblicherweise nicht in DL-Sprachen notiert sind. Eine TBox ist das mathematische Modell einer Wissensbasis.

selbst definiert ist — das folgende Beispiel ist eine Variation aus [17]. Durch die Konzeptdefinition

$$\text{Horse} \equiv \text{Mammal} \sqcap (\exists^{=2}\text{parents} \,.\, \text{Horse})$$

werden Pferde als Säugetiere beschrieben, deren Eltern Pferde sind. Das zu definierende Konzept kommt auch in der Definition des Konzeptes selbst vor und bildet einen Zyklus[3]. Das Vorkommen von Zyklen in TBoxen wirft eine Reihe von Problemen auf; zum einen wird das Subsumtionsproblem i.d.R. schwerer als im azyklischen Fall. Zum anderen muss man in Falle zyklischer TBoxen verschiedene Semantiken in Betracht ziehen, wodurch bei der Berechnung von Subsumtion jeweils verschiedene Ergebnisse zustande kommen. Neben der *deskriptiven Semantik* auch die Semantik des *größten Fixpunkts* (*gfp*-Semantik) und die Semantik des *kleinsten Fixpunkts* (*lfp*-Semantik). Die Begriffe gehen auf Bernhard Nebel zurück, der sie in seiner Untersuchung über terminologische Zyklen in [17] eingeführt hat. Da in der DL-Sprache $\mathcal{EL}$ in *lfp*-Modellen alle zyklischen Definitionen als die leere Menge interpretiert werden, ist diese Semantik von geringem Interesse und spielt in unserem Zusammenhang gar keine Rolle. Mit *gfp*-Semantik werden nur die größtmöglichen Modelle in Betracht gezogen, mit deskriptiver Semantik hingegen alle Modelle. Zur Semantik von terminologischen Zyklen siehe auch [2]. Für die Sprache $\mathcal{EL}$ wurde gezeigt, dass das Subsumtionsproblem für zyklische wie für azyklische TBoxen sowohl mit *lfp*-Semantik, deskriptiver Semantik als auch mit *gfp*-Semantik in polynomieller Zeit zu lösen ist [3], was im Vergleich mit anderen DLs ein recht erstaunliches Ergebnis ist [1, 2].

Eine TBox, die s.g. GCIs (engl.: *general concept inclusion axioms*) enthält, nennt man generelle TBox. Ein klassisches Beispiel [14] für ein GCI ist

$$\begin{aligned} &\mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation.Stomach} \\ &\quad \sqsubseteq \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation.}(\mathsf{Lining} \sqcap \exists \mathsf{isPartOf.Stomach}); \end{aligned}$$

es drückt aus, dass ein Magengeschwür immer ein Geschwür der Magenschleimhaut ist. Hier zeigt sich auch bereits eine der Anwendungsmöglichkeiten von

[3] Natürlich machen nicht alle zyklischen Definitionen Sinn; die Konzepte : 'Male-Human $\equiv$ Man $\sqcap$ Human ' sowie 'Man $\equiv$ Human $\sqcap$ Male-Human ' bilden zusammen eine *komponentenweise zyklische Definition*, die unsinnig ist. Das Beispiel und die Bezeichnung '*komponentenweise zyklische Definition*' stammen von Bernhard Nebel [17].

GCIs in Wissensbasen: bestehende Konzeptbeschreibungen können durch GCIs nachträglich mit zusätzlicher Information versehen werden, was für die Wartung und Erweiterung von bestehenden Wissensbasen von Bedeutung sein kann. Ein Beispiel für eine Wissensbasis, in der von GCIs Gebrauch gemacht wird, findet sich im GALEN-Projekt [19]. Wiederum gilt, Subsumtion für generelle TBoxen ist im Allgemeinen schwieriger als für einfache TBoxen und wiederum gilt dies nicht für $\mathcal{EL}$. In [9] wurde ein polynomieller Algorithmus zur Berechnung von Subsumtion für generelle $\mathcal{EL}$-TBoxen[4] mit deskriptiver Semantik vorgestellt. Man beachte, dass es beim Vorhandensein von GCIs keinen Sinn macht, eine andere als die deskriptive Semantik in Betracht zu ziehen.

In der Familie der Beschreibungslogiken gibt es Sprachen mit großer Ausdrucksstärke [15], die trotz der schlechten *'worst case'* Komplexität ihrer Inferenzverfahren ein akzeptables Verhalten in realen Anwendungen aufweisen [16]. Ein Grund, warum die Untersuchung ausdrucksschwacher Beschreibungslogiken trotzdem von Interesse ist, liegt — abgesehen davon, dass es in einigen Fällen für praktische Belange anscheinend ausreichend ist, sub-boolesche Sprachen zu verwenden [12, 19] — an der Bedeutung von *Nicht-Standard-Inferenzverfahren*, die eine wichtige Rolle bei der Konstruktion und Wartung von großen Terminologien spielen. Solche Nicht-Standardinferenzen sind z.B. das *speziellste Konzept bezüglich zweier Objekte*[5] (msc von engl.: *most specific concept*) und das *kleinste gemeinsame Oberkonzept* (lcs von engl.: *least common subsumer*) bezüglich zweier Konzepte. Die lcs- und msc-Inferenzen benötigt man, wenn bei der Konstruktion von TBoxen eine *bottom-up*-Strategie verfolgt wird, was oftmals die von Ingenieuren vorgezogene Methode ist [8, 20]. Des weiteren kann man mit dem lcs in der Hierarchie der Konzepte der TBox weitere Ebenen einfügen. Die lcs-Inferenz aber macht nur in Beschreibungslogiken wie z.B. $\mathcal{EL}$ ohne volle Negation und Disjunktion Sinn, da sie ansonsten trivial ist und keine relevante Information durch ihre Berechnung gewonnen würde. Für die Beschreibungslogik $\mathcal{EL}$ wurde nachgewiesen, dass das lcs für zyklische TBoxen immer existiert

[4] D.h. sogar für $\mathcal{ELH}$-TBoxen — das Verfahren behandelt auch Rollenaxiome korrekt.

[5] *Objekte* sind hier Entitäten aus der Domäne, die durch Konzeptbeschreibungen in der TBox klassifiziert wird. Das msc-Inferenzverfahren kann dazu benutzt werden, um aus wenigstens zwei solcher Objekten der realen Welt eine neue Konzeptdefinition in der TBox zu generieren

und in polynomieller Laufzeit berechnet werden kann, wenn man ausschließlich *gfp*-Modelle in Betracht zieht [7]; mit deskriptiver Semantik ist hingegen nicht garantiert, dass das lcs existiert [6].

Das Resultat aus [6] zusammen mit der Tatsache, dass GCIs nur mit deskriptiver Semantik sinnvoll untersucht werden können, verhindert also zunächst einmal die Verwendung von GCIs in TBoxen, wenn man auch die Nicht-Standardinferenz lcs verwenden möchte. Um dieses Problem zu lösen, wurden ein Art hybrider $\mathcal{EL}$-TBoxen konzipiert, in der sowohl zyklische Definitionen vorkommen, die mit *gfp*-Semantik betrachtet werden, als auch s.g. *primitive* GCIs , die mit deskriptiver Semantik betrachtet werden. Wir zeigen, dass diese hybriden $\mathcal{EL}$-TBoxen in polynomieller Zeit unter Wahrung der *gfp*-Semantik in einfache $\mathcal{EL}$-TBoxen umgewandelt werden können. Das erlaubt es, diese hybriden TBoxen wahlweise als generelle TBoxen zu betrachten und mit deskriptiver Semantik darüber zu schließen, oder aber — nach der Transformation — als zyklische TBox und sie mit *gfp*-Semantik auf Subsumtion und lcs-Inferenz zu untersuchen.

Im nächsten Kapitel werden wir im Abschnitt 2.1 zunächst eine formalen Definition der Sprache $\mathcal{EL}$ und die übliche Semantik für EL-Konzepte angeben. Im Abschnitt 2.2 kommen wir dann zur *gfp*-Semantik und der Subsumtion mit verschiedenen Semantiken, in Abschnitt 2.3 führen wir GCIs und Subsumtion für TBox mit GCIs ein, um in in Abschnitt 2.4 die hybriden TBoxen vorstellen und eine Semantik für die Subsumtion bzgl. solcher TBoxen angeben. Außerdem werden wir anhand eines Beispiels das Problem aufzeigen, welches durch das Vorkommen von primitiven GCIs entsteht. In Kapitel 3 entwickeln wir dann in das Transformationsverfahren und beweisen, dass es auch korrekt und vollständig ist. In Kapitel 4 schließlich werden wir eine prototypische Implementierung in LISP vorstellen, die auf bestehenden Implementierungen für das Subsumtionsproblem von generellen TBoxen und zyklischen $\mathcal{EL}$-TBoxen mit *gfp*-Semantik [22] aufbaut und die Resultate von Funktionstests vorstellen.

Kapitel 2

Syntax, Semantiken und Subsumtionen

In diesem Kapitel werden wir zunächst die Syntax für $\mathcal{EL}$ und die — deskriptive — Semantik für $\mathcal{EL}$-Konzepte definieren, sowie den Begriff der TBox formal einführen. Im Abschnitt 2.2 werden wir die *gfp*-Semantik vorstellen und insbesondere Subsumtion bezüglich deskriptiver Semantik und *gfp*-Semantik unterscheiden. Der Abschnitt 2.3 behandelt die generellen TBoxen und das Verfahren zur Entscheidung der Subsumtion bezüglich genereller $\mathcal{EL}$-TBoxen mit deskriptiver Semantik. Alle in den ersten drei Abschnitten eingeführten Definitionen stammen entweder aus [3] oder [9]. Im Abschnitt 2.4 werden wir schließlich hybride TBoxen mit zyklischen Definitionen und primitiven GCIs sowie eine Semantik für Subsumtion bezüglich dieser TBoxen mit *gfp*-Semantik vorstellen. Abschließend erläutern wir anhand eines Beispiels den Effekt der primitiven GCIs auf die Subsumtionsbeziehungen einer TBox.

2.1 Die Beschreibungslogik $\mathcal{EL}$

Syntax und deskriptive Semantik von $\mathcal{EL}$

Die Beschreibungslogik $\mathcal{EL}$ ist induktiv definiert mittels einer Menge von Konzeptnamen $\mathbf{N}_C$, einer Menge von Rollennamen $\mathbf{N}_R$ und einer Menge von Konstruktoren, die es gestatten, aus einfachen Konzepten komplexere zu bilden.

Konstruktor	**Syntax**	**Semantik**
Topkonzept	$\top$	$\Delta^{\mathcal{I}}$
Konjunktion	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Existenzielle Restriktion	$\exists r.C$	$\{\, x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}} \text{ und } y \in C^{\mathcal{I}}\}$

Tabelle 2.1: Syntax und Semantik für $\mathcal{EL}$-Konzepte

In $\mathcal{EL}$ stehen uns das Topkonzept ‚$\top$' und für zwei Konzepte C und D und eine Rolle $r \in \mathbf{N}_R$ die Konjunktion ‚$C \sqcap D$' und die existenzielle Restriktion ‚$\exists r.C$' zur Verfügung, um weitere $\mathcal{EL}$-Konzepte zu bilden. Die Semantik von $\mathcal{EL}$-Konzeptbeschreibungen ist durch eine Interpretation $\mathcal{I} = \left(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\right)$ gegeben, wobei die Domäne $\Delta^{\mathcal{I}}$ eine nicht-leere Menge ist und $\cdot^{\mathcal{I}}$ eine Funktion, die jeden Konzeptnamen $P \in \mathbf{N}_C$ auf eine Menge $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ abbildet und jeden Rollennamen $r \in \mathbf{N}_R$ auf eine binäre Relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Die Erweiterung von $\cdot^{\mathcal{I}}$ auf beliebige Konzeptbeschreibungen ist induktiv definiert und in Tabelle 2.1 angegeben.

□

$\mathcal{EL}$-TBoxen

Eine $\mathcal{EL}$-Terminologie oder kurz $\mathcal{EL}$-TBox $\mathcal{T}$ ist eine endliche Menge von Konzeptdefinitionen der Form $A \equiv D$. Dabei ist A ein Name aus $\mathbf{N}_C$ und wir nennen A ein in $\mathcal{T}$ definiertes Konzept; die rechte Seite einer Konzeptdefinition $A \equiv D$ ist eine Konzeptbeschreibung D aus $\mathcal{EL}$. Die Menge aller durch eine TBox $\mathcal{T}$ definierten Konzepte bezeichnen wir mit $\mathbf{N}_{def}^{\mathcal{T}}$ oder nur $\mathbf{N}_{def}$, wenn klar ist auf welche TBox wir uns beziehen. Üblicherweise werden wir für die Bezeichnung definierter Konzepte große Buchstaben vom Anfang des lateinischen Alphabets verwenden, also z.B. $\{A, B, A_1, A_2, \ldots\}$. Alle anderen Konzeptnamen, die in einer TBox $\mathcal{T}$ vorkommen, heißen *primitive Konzepte*; die Menge aller primitiven Konzepte einer TBox $\mathcal{T}$ heißt $\mathbf{N}_{prim}^{\mathcal{T}}$ oder — bei Eindeutigkeit der Bezeichnung — nur $\mathbf{N}_{prim}$. Üblicherweise werden wir für die Bezeichnung primitiver Konzepte große Buchstaben wie P, Q, R, $P_1, \ldots$ verwenden. Es gilt $\mathbf{N}_{def}^{\mathcal{T}} \cap \mathbf{N}_{prim}^{\mathcal{T}} = \emptyset$ und jedes Konzept $A \in \mathbf{N}_{def}$ ist nur genau einmal auf der linken Seite einer Konzeptdefinition in $\mathcal{T}$ enthalten. Die Menge aller Rollennamen, die in einer

TBox $\mathcal{T}$ vorkommen, nennen wir $\mathbf{N}_r^{\mathcal{T}}$ bzw. $\mathbf{N}_r$ und es gilt $\mathbf{N}_r^{\mathcal{T}} \subseteq \mathbf{N}_R$. In der definierenden Konzeptbeschreibung ‚D' dürfen alle Konzeptnamen aus $\mathbf{N}_{prim}^{\mathcal{T}}$ und $\mathbf{N}_{def}^{\mathcal{T}}$ und sogar das durch D definierte Konzept A selber auftauchen. Definitionen, deren rechte Seite direkt oder indirekt auf ihre linke Seite Bezug nehmen, werden *zyklisch* genannt. Eine Interpretation $\mathcal{I}$ heißt *Modell* für eine TBox $\mathcal{T}$, wenn für jede Konzeptdefinitionen $(A \equiv D) \in \mathcal{T}$ gilt, dass $A^{\mathcal{I}} = D^{\mathcal{I}}$ — wir schreiben dann $\mathcal{I} \models \mathcal{T}$.

□

Definition 2.1.1 Die Größe einer TBox $\mathcal{T}$ — in Zeichen $||\mathcal{T}||$ — ergibt sich aus der Summe der Größe aller in ihr enthaltenen Konzeptdefinitionen. Die Größe einer Konzeptdefinition $||A \equiv D||$ errechnet sich rekursiv sich wie folgt: Konzeptnamen haben die Größe 1, für eine Konzeptdefinition gilt $||A \equiv D|| = 1 + ||D||$, für Konjunktionen $||(C \sqcap D)|| = 1 + ||C|| + ||D||$ und für Existenzrestriktionen $||\exists r.C|| = 1 + ||C||$.

2.2 *gfp*-Semantik

Die bisher vorgestellte Semantik für — möglicherweise zyklische — $\mathcal{EL}$-TBoxen heißt nach B. Nebel *deskriptive* Semantik [17]. Vor der formale Definition der *gfp*-Semantik soll anhand eines Beispiels aus [3] der Effekt der Interpretation mit *gfp*-Semantik erläutert werden. Betrachten wir die folgende $\mathcal{EL}$-TBox mit einer zyklischen Konzeptdefinition:

$$\mathsf{Inode} \equiv \mathsf{Node} \sqcap \exists \mathsf{edge}.\mathsf{Inode}$$

sowie die folgende Interpretation $\mathcal{J}$ für das primitive Konzept ‚Node' und die binäre Rolle ‚edge'

$$\begin{aligned} \Delta^{\mathcal{J}} &:= \{m_1, m_2, m_3 \ldots\} \cup \{n_0\} \\ \mathsf{Node}^{\mathcal{J}} &:= \Delta^{\mathcal{J}} \\ \mathsf{edge}^{\mathcal{J}} &:= \{(m_i, m_{m+1}) \mid i \geq 0\} \cup \{(n_0, n_0)\} \end{aligned}$$

Die Interpretation $\mathcal{J}$ kann auf vier verschiedene Arten zu einer Modell für die TBox erweitert werden: ‚Inode' kann als die Menge $\emptyset$, $\{n_0\}$, $\{m_1, m_2, m_3 \ldots\}$ oder $\{m_1, m_2, m_3 \ldots\} \cup \{n_0\}$ interpretiert werden. Mit deskriptiver Semantik

sind alle Modelle zulässig, mit *gfp*-Semantik kann das Konzept ‚Inode' nur als die Menge $\{m_1, m_2, m_3 \ldots\} \cup \{n_0\}$ interpretiert werden, was in diesem Fall auch der intendierten Bedeutung der Konzeptdefinition entspricht.

Sei im folgenden durch $\mathcal{T} = \{A_1 \equiv D_1, \ldots, A_k \equiv D_k\}$ eine $\mathcal{EL}$-TBox festgelegt mit den definierten Konzepten $\mathbf{N}_{def} = \{A_1, \ldots, A_k\}$, den primitiven Konzepten $\mathbf{N}_{prim}$ und der Menge $\mathbf{N}_r$ der Rollen, die in $\mathcal{T}$ vorkommen.

DEFINITION 2.2.1 Eine Interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ heißt *primitive Interpretation* für $\mathcal{T}$, wenn durch $\mathcal{J}$ jedes primitive Konzept $P \in \mathbf{N}_{prim}$ auf eine Teilmenge $P^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}}$ abgebildet wird und jede Rolle $r \in \mathbf{N}_r$ auf eine binäre Relation $r^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$, die Konzepte $A_1 \ldots A_k$ jedoch nicht interpretiert werden. □

DEFINITION 2.2.2 Eine Interpretation $\mathcal{I}$ für $\mathcal{T}$ basiert auf einer primitiven Interpretation $\mathcal{J}$ für $\mathcal{T}$ gdw. $\mathcal{I}$ und $\mathcal{J}$ die gleiche Domäne haben und in der Interpretation von $\mathbf{N}_{prim}$ und $\mathbf{N}_r$ übereinstimmen. Zu einer festen primitiven Definition $\mathcal{J}$ für $\mathcal{T}$ sind die Interpretationen $\mathcal{I}$, die auf $\mathcal{J}$ basieren, eindeutig durch ein Tupel $(A_1^{\mathcal{I}}, \ldots, A_k^{\mathcal{I}})$ von Interpretationen der definierten Konzepte aus $\mathbf{N}_{def}$ bestimmt.Die Menge aller Interpretationen $\mathcal{I}$ die auf $\mathcal{J}$ basieren, bezeichnen wir als $Int(\mathcal{J}) = \{\ \mathcal{I} \mid \mathcal{I}$ basiert auf $\mathcal{J}\}$.

□

Auf $Int(\mathcal{J})$ ist eine partielle Ordnung $\preceq_{\mathcal{J}}$ definiert: wenn $\mathcal{I}_1,\ \mathcal{I}_2 \in Int(\mathcal{J})$, dann gilt:

$$\mathcal{I}_1 \preceq_{\mathcal{J}} \mathcal{I}_2 \quad \Leftrightarrow \quad \forall i \in \{1 \ldots k\} :\ A_i^{\mathcal{I}_1} \subseteq A_i^{\mathcal{I}_2}$$

Durch das Tupel $\langle Int(J), \preceq_{\mathcal{J}} \rangle$ ist ein *vollständiger Verband* definiert, d.h. zu jeder Teilmenge von $Int(\mathcal{J})$ existiert eine größte untere Grenze (*glb*) und eine kleinste obere Grenze (*lub*) und Tarskis Fixpunkttheorem [23] kann angewendet werden. Das Fixpunkttheorem von Tarski besagt, dass jede monotone Funktion $\mathcal{O}\colon Int(\mathcal{J}) \to Int(\mathcal{J})$ einen größten und einen kleinsten Fixpunkt besitzt. In [3], Abschnitt 2, Definition 2 wurde eine monotone Funktion $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ definiert mit der Eigenschaft, dass die Menge der Fixpunkte von $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ genau die Menge der Interpretationen $\mathcal{I} \in Int(\mathcal{J})$ für $\mathcal{T}$ umfasst, für die gilt: $\mathcal{I} \models \mathcal{T}$. Das heißt, $\mathcal{I} \in Int(\mathcal{J})$ ist genau dann ein Fixpunkt von $\mathcal{O}_{\mathcal{T},\mathcal{J}}$, wenn $\mathcal{I}$ ein Modell für $\mathcal{T}$ ist.

DEFINITION 2.2.3 Die TBox $\mathcal{T}$ induziert eine Funktion $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ auf $Int(\mathcal{J})$ so dass für zwei Interpretationen $\mathcal{I}_1,\ \mathcal{I}_2 \in Int(\mathcal{J})$ gilt:

$$\mathcal{O}_{\mathcal{T},\mathcal{J}}(\mathcal{I}_1) = \mathcal{I}_2 \quad gdw. \quad A_i^{\mathcal{I}_2} = D_i^{\mathcal{I}_1} \text{ für alle } i \in \{1 \ldots k\}.$$

□

Wir sind nun soweit eine formale Definition der Semantik des größten Fixpunktes (*gfp*-Semantik) und der Subsumtion bezüglich verschiedener Semantiken zu geben.

DEFINITION 2.2.4 Eine Interpretation $\mathcal{I}$ für eine TBox $\mathcal{T}$ heißt *gfp*-Modell für $\mathcal{T}$ gdw. es eine primitive Interpretation $\mathcal{J}$ für $\mathcal{T}$ gibt, so dass $\mathcal{I} \in Int(\mathcal{J})$ der größte Fixpunkt für $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ ist.

□

In Bezug auf die *gfp*-Semantik sind nur *gfp*-Modelle zulässig.

DEFINITION 2.2.5 Seien A und B zwei Konzepte aus $\mathcal{T}$. Das Konzept A wird von B bezüglich *deskriptiver* Semantik subsumiert — in Zeichen $A \sqsubseteq_{\mathcal{T}} B$ — gdw. $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ in allen Modellen $\mathcal{I}$ von $\mathcal{T}$ gilt.

Das Konzept A wird von B bezüglich *gfp*-Semantik subsumiert — in Zeichen $A \sqsubseteq_{\mathcal{T},gfp} B$ — gdw. $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ in allen *gfp*-Modellen $\mathcal{I}$ von $\mathcal{T}$ gilt.

□

Es soll hier ausdrücklich bemerkt werden, dass folgende Implikation eine unmittelbare Konsequenz aus der Definition von $\sqsubseteq_{\mathcal{T},gfp}$ bzw. $\sqsubseteq_{\mathcal{T}}$ ist:

$$A \sqsubseteq_{\mathcal{T}} B \quad \Rightarrow \quad A \sqsubseteq_{\mathcal{T},gfp} B.$$

Jetzt sollen noch einige Definitionen aus [3] wiedergegeben werden, die wir später für den Beweis 3.2.2 im Abschnitt 3.2 brauchen.

DEFINITION 2.2.6 Für eine absteigende Kette von Interpretationen $\mathcal{I}_0 \succeq \mathcal{I}_1 \succeq \mathcal{I}_2 \succeq \mathcal{I}_3 \ldots$ aus $Int(\mathcal{J})$ ist die Interpretation $\mathcal{I}$ mit $A_i^{\mathcal{I}} = \bigcap_{j \geq 0} A_i^{\mathcal{I}_j}$ die kleinste untere Grenze (*glb*); die Funktion $\mathcal{O}_{\mathcal{T},\mathcal{J}} : Int(\mathcal{J}) \to Int(\mathcal{J})$ ist *abwärts stetig*, gdw.

$$O_{\mathcal{T},\mathcal{J}}(glb(\{\mathcal{I}_i \mid i \geq 0\})) = glb(\{O_{\mathcal{T},\mathcal{J}}(\mathcal{I}_i) \mid i \geq 0\})$$

□

Leider ist die Funktion $\mathcal{O}_{\mathcal{T},\mathcal{J}} : Int(\mathcal{J}) \rightarrow Int(\mathcal{J})$ nicht unbedingt abwärts stetig, wie in [3] gezeigt wurde. Eine Konsequenz daraus ist, dass der größte Fixpunkt nicht unbedingt mehr durch einfache ω-Iteration gefunden werden kann. Mit ‚ω' bezeichnen wir die erste unendliche Ordinalzahl.

DEFINITION 2.2.7 Sei $\mathcal{T}$ eine $\mathcal{EL}$-TBox mit $\mathbf{N}_{def} = \{A_1 \ldots A_k\}$, $\mathcal{J}$ eine primitive Interpretation und $\mathcal{I}_{top}$ die größte Interpretation für $\mathcal{T}$ aus $Int(\mathcal{J})$, d.h. $A_i^{\mathcal{I}_{top}} = \Delta^{\mathcal{J}}$ für alle $A_i \in \mathbf{N}_{def}$ mit $1 \leq i \leq k$. Für jede Ordinalzahl α ist $\mathcal{I}^{\downarrow\alpha}$ wie folgt definiert:

- $\mathcal{I}^{\downarrow\alpha} := \mathcal{I}_{top}$ wenn $\alpha = 0$;
- $\mathcal{I}^{\downarrow\alpha+1} := \mathcal{O}_{\mathcal{T};\mathcal{J}}(\mathcal{I}^{\downarrow\alpha})$;
- $\mathcal{I}^{\downarrow\alpha} := glb(\{\mathcal{I}^{\downarrow\beta} \mid \beta < \alpha\})$, wenn α eine Grenzwert-Ordinale ist.

□

Für die Berechnung der *gfp*-Subsumtion für $\mathcal{EL}$-TBoxen mit zyklischen Definitionen wird ein $\mathcal{EL}$-TBox zunächst in in eine Normalform gebracht und anschließend als ein Graph dargestellt, auf dem dann die *größte Simulation* berechnet wird.

DEFINITION 2.2.8 Ein TBox $\mathcal{T}$ hat *Normalform für Graphendarstellung* gdw. die in $\mathcal{T}$ enthaltenen Konzeptdefinitionen die Form

$$A \equiv P_1 \sqcap \ldots \sqcap P_n \sqcap \exists r_1.B_1 \sqcap \ldots \sqcap \exists r_m.B_m$$

haben, wobei $P_i \in \mathbf{N}_{prim} \quad \forall i \in \{1 \ldots n\}$ und $B_i \in \mathbf{N}_{def} \quad \forall i \in \{1 \ldots m\}$.

□

In [3] wurde gezeigt, dass sich jede $\mathcal{EL}$-TBox in polynomieller Zeit in eine normalisierte TBox umwandeln läßt. Unter der Voraussetzung, dass $\mathcal{T}$ normalisiert ist, kann eine TBox $\mathcal{T}$ auch als Graph aufgefasst werden:

DEFINITION 2.2.9 Ein *$\mathcal{EL}$-Beschreibungsgraph* ist ein beschrifteter Graph $\mathcal{G} = (V, E, L)$ wobei

- V eine Menge von Knoten ist,

- $E \subseteq V \times \mathbf{N}_R \times V$ eine Menge von Kanten ist, die jeweils mit Rollennamen aus $\mathbf{N}_R$ beschriftet sind und
- $L\colon V \to 2^{\mathbf{N}_{prim}}$ eine Funktion ist, die je einen Knotennamen mit einer Menge von primitiven Konzepten beschriftet.

□

Jede $\mathcal{EL}$-TBox $\mathcal{T}$ kann in einen $\mathcal{EL}$-Beschreibungsgraphen $\mathcal{G}_\mathcal{T} = (\mathbf{N}_{def}, E_\mathcal{T}, L_\mathcal{T})$ umgewandelt werden. Dabei gilt, die Menge der Knoten ist genau die Menge der definierten Konzepte $\mathbf{N}_{def}$ und wenn ein Konzept A in $\mathcal{T}$ definiert ist als

$$A \equiv P_1 \sqcap \ldots \sqcap P_n \sqcap \exists r_1.B_1 \sqcap \ldots \sqcap \exists r_m.B_m,$$

dann ist $L_\mathcal{T}(A) = \{P_1, \ldots, P_n\}$ und $(A, r_j, B_j) \in E_\mathcal{T}$ für alle $1 \leq j \leq m$. Ein Beispiel für einen $\mathcal{EL}$-Beschreibungsgraphen ist in Figur 2.1 auf Seite 16 gezeigt.

Genauso kann jede primitive Interpretation $\mathcal{J}$ in einen $\mathcal{EL}$-Beschreibungsgraphen $\mathcal{G}_\mathcal{J} = (\Delta^\mathcal{J}, E_\mathcal{J}, L_\mathcal{J})$ umgewandelt werden. Dann gilt, die Menge der Knoten ist $\Delta^\mathcal{J}$; für die Kantenmenge gilt $E_\mathcal{J} = \{(x, r, y) \mid (x, y) \in r^\mathcal{J}\}$ und $L_\mathcal{J}(x) = \{P \in \mathbf{N}_{prim} \mid x \in P^\mathcal{J}\}$ für alle $x \in \Delta^\mathcal{J}$. Da man umgekehrt auch jeden $\mathcal{EL}$-Beschreibungsgraphen als primitive Interpretation auffassen kann, lässt sich aus dem $\mathcal{EL}$-Beschreibungsgraphen $\mathcal{G}_\mathcal{T}$ für eine TBox $\mathcal{T}$ eine *kanonische* primitive Interpretation $\mathcal{K} = (\Delta^\mathcal{K}, \cdot^\mathcal{K})$ gewinnen mit $\Delta^\mathcal{K} = \mathbf{N}_{def}$ und $(A, B) \in r^\mathcal{K}$ gdw. $(A, r, B) \in \mathcal{EL}_\mathcal{T}$ (gdw. $\exists r.B$ in der Definition von A vorkommt) sowie $A \in P^\mathcal{K}$ gdw. $P \in L_\mathcal{T}(A)$. Wir werden von einer solchen kanonischen Interpretation später Gebrauch machen.

Um für eine $\mathcal{EL}$-TBox $\mathcal{T}$ die *gfp*-Subsumtion zu berechnen, wird aus $\mathcal{T}$ der Graph $\mathcal{G}_\mathcal{T}$ erzeugt und auf dem Graphen die größte *Simulationsrelation* $Z\colon \mathcal{G}_\mathcal{T} \overset{\sim}{\to} \mathcal{G}_\mathcal{T}$ berechnet, woraus sich dann direkt die Subsumtionsbeziehungen ablesen lassen: genau dann, wenn ein Paar $(B, A) \in Z$ für zwei Knoten A und B, gilt auch $A \sqsubseteq_{\mathcal{T},gfp} B$.

Definition 2.2.10 Wenn $\mathcal{G}_1 = (V_1, E_1, L_1)$ und $\mathcal{G}_2 = (V_2, E_2, L_2)$ zwei $\mathcal{EL}$-Beschreibungsgraphen sind, dann ist die binäre Relation $Z \subseteq V_1 \times V_2$ eine Simulation von $\mathcal{G}_1$ nach $\mathcal{G}_2$ gdw. gilt

1. $(v_1, v_2) \in Z$ impliziert $L_1(v_1) \subseteq L_2(v_2)$ und

2. wenn $(v_1, v_2) \in Z$ und $(v_1, r, v_1') \in E_1$, dann existiert ein Knoten $v_2' \in V_2$ mit $(v_2, r, v_2'') \in E_2$ und $(v_1', v_2') \in Z$.

□

Für alle weiteren Details sei auf [3], Abschnitt 3 verwiesen. Ein effizienter Algorithmus zur Berechnung von Simulationen auf beschrifteten Graphen wurde in [13] vorgestellt — eine Implementation des Algorithmus in [22].

2.3 Subsumtion für $\mathcal{EL}$-TBoxen mit GCIs

In diesem Abschnitt werden die generellen TBoxen formal vorgestellt und das Verfahren zur Berechnung von Subsumtion mit deskriptiver Semantik bezüglich dieser TBoxen informell erläutert, sowie die für uns relevanten Definition der *Normalform* von generellen $\mathcal{EL}$-TBoxen sowie der *Subsumtionsmenge* wiedergegeben.

Definition 2.3.1 TBoxen mit GCIs
Seien C und D zwei beliebige $\mathcal{EL}$-Konzepte. Ein GCI ist ein Axiom der Form $C \sqsubseteq D$. Eine Definition ist ein Axiom der Form $C \equiv D$. Definitionen können als Paar von GCIs aufgefasst werden ($C \sqsubseteq D$ und $D \sqsubseteq C$). Eine endliche Menge von GCIs und/oder Definitionen, heißt *generelle* TBox. Eine Interpretation $\mathcal{I}$ ist ein Modell für eine generelle TBox $\mathcal{V}$ gdw. für jedes GCI $C \sqsubseteq D$ aus $\mathcal{V}$ gilt $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ und für jede Definition $C \equiv D$ aus $\mathcal{T}$ $C^{\mathcal{I}} = D^{\mathcal{I}}$.

□

Bemerkung: In generellen $\mathcal{EL}$-TBoxen gibt es die Unterscheidung zwischen primitiven und definierten Konzepten nicht. Die Definition der Subsumtion von Konzepten aus einer generellen TBox entspricht der für allgemeine TBoxen mit deskriptiver Semantik — siehe Definition 2.2.5. Wir benutzen hier in Abweichung vom gewöhnlichen Gebrauch das Symbol ‚$\mathcal{V}$ ‘, um eine generellen $\mathcal{EL}$-TBox zu bezeichnen, damit auch symbolisch zwischen den zwei Arten von $\mathcal{EL}$-TBoxen unterschieden wird.

Der in [9] entwickelte Algorithmus zur Subsumtionsberechnung über $\mathcal{EL}$-TBoxen mit GCIs bringt die $\mathcal{EL}$-TBox zunächst in eine Normalform und berechnet dann für jeden Konzeptnamen P, der in der TBox definiert ist, s.g.

Subsumtionsmengen die alle Konzeptnamen enthalten, von denen P subsumiert wird; daraus können dann direkt die Subsumtionsbeziehungen abgelesen werden.

Definition 2.3.2 Normalform für primitive TBoxen
Eine generelle $\mathcal{EL}$-TBox $\mathcal{V}$ hat *Normalform zur Subsumtionsmengenberechnung*, gdw. $\mathcal{V}$ nur GCIs der Form

$$\begin{aligned} P &\sqsubseteq Q \\ P &\sqsubseteq \exists r.Q \\ \exists r.Q &\sqsubseteq P \\ P \sqcap Q &\sqsubseteq R \end{aligned}$$

enthält, mit $\{P, Q, R\} \subseteq \mathbf{N}_{prim}$ und $r \in \mathbf{N}_r$.

□

Es wurde gezeigt, dass sich jede generelle $\mathcal{EL}$-TBox $\mathcal{V}$ in polynomieller Zeit normalisieren läßt und die Größe von $\mathcal{V}$ dadurch nur linear anwächst [9].

Definition 2.3.3 Subsumtionsmengen
Sei $\mathcal{V}$ eine generelle $\mathcal{EL}$-TBox und P und Q Konzeptnamen, die in GCIs aus $\mathcal{V}$ vorkommen. Eine Menge $S_*(P)$ für einen Konzeptnamen P heißt *Subsumtionsmenge für* P gdw. $Q \in S_*(P) \quad \Leftrightarrow \quad P \sqsubseteq_{\mathcal{V}} Q$

□

Die Subsumtionsmengen können in polynomieller Zeit abhängig von der Größe der generellen TBox berechnet werden — auch für dieses Verfahren existiert eine Implementation, die in [22] vorgestellt wurde.

Bemerkung: das Verfahren zur Berechnung von Subsumtionsmengen funktioniert auch für zyklische TBoxen (ohne GCIs) — man kann die Konzeptdefinition $A \equiv \widehat{D}$ als ein Paar von GCIs $A \sqsubseteq \widehat{D}$ und $\widehat{D} \sqsubseteq A$ auffassen.

2.4 Hybride $\mathcal{EL}$-TBoxen mit primitiven GCIs

Dieser Abschnitt behandelt schließlich eine neue Form von $\mathcal{EL}$-TBoxen, für die wir in Kapitel 3 ein Verfahren zur Berechnung von *gfp*-Subsumtion angeben werden.

DEFINITION 2.4.1 $\mathcal{EL}$-TBoxen mit primitiven GCIs
Sei $\mathcal{T}$ eine beliebige $\mathcal{EL}$-TBox und C und D zwei beliebige $\mathcal{EL}$-Konzepte, in denen aber *keine* Konzeptnamen aus $\mathbf{N}_{def}$ vorkommen. Ein primitives GCI für $\mathcal{T}$ ist ein Axiom der Form $C \sqsubseteq D$. Eine endliche Menge $\mathcal{V}$ von primitiven GCIs, die mit einer TBox $\mathcal{T}$ assoziiert ist, nennen wir *primitive generelle TBox für* $\mathcal{T}$. Primitive GCIs schränken die Menge der primitiven Interpretationen $\mathcal{J}$ für $\mathcal{T}$ ein; als zulässig gelten primitive Interpretationen für $\mathcal{T}$ nur dann, wenn sie auch Modelle für $\mathcal{V}$ sind — also $\mathcal{J} \models \mathcal{V}$ gilt. Eine primitive Interpretation $\mathcal{J}$ für eine TBox $\mathcal{T}$ mit assoziierter primitiver TBox $\mathcal{V}$ ist also genau dann zulässig, wenn für alle primitiven GCIs $C \sqsubseteq D \in \mathcal{V}$ gilt: $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$, d.h. $\mathcal{J} \models \mathcal{V}$. TBoxen mit assoziierten primitiven TBoxen nennen wir hybride TBoxen.
□

Wir definieren noch eine Hilfsfunktion, die wir in einer angepassten Definition für Subsumtion in hybriden TBoxen brauchen, genauso wie im Beweis von Theorem 3.2.2 (siehe Abschnitt 3.2).

DEFINITION 2.4.2
Mit $\mathsf{gfp}(\mathcal{T}, \mathcal{J})$ ist die Funktion bezeichnet, die für eine TBox $\mathcal{T}$ und eine primitive Interpretation $\mathcal{J}$ für $\mathcal{T}$ genau das *gfp*-Model aus $Int(\mathcal{J})$ zurück liefert.
□

Unsere erweiterte Definition macht nun allerdings auch eine neue Definition von *gfp*-Subsumtion[1] erforderlich:

DEFINITION 2.4.3 *gfp*-Subsumtion bezüglich hybrider $\mathcal{EL}$-TBoxen
Sei $\mathcal{T}$ eine TBox mit assoziierter primitiver TBox $\mathcal{V}$ und A und B zwei definierte Konzepte aus $\mathbf{N}_{def}$. Das Konzept A wird von B bezüglich der hybriden TBox $(\mathcal{T} \cup \mathcal{V})$ mit *gfp*-Semantik subsumiert — in Zeichen $A \sqsubseteq_{\mathcal{T},\mathcal{V},gfp} B$ — gdw. für alle primitiven Interpretationen $\mathcal{J}$ mit $\mathcal{J} \models \mathcal{V}$ gilt $A^{\mathsf{gfp}(\mathcal{T},\mathcal{J})} \subseteq B^{\mathsf{gfp}(\mathcal{T},\mathcal{J})}$.
□

D.h. die Menge der primitiven Interpretationen ist im Fall einer primitiven TBox $\mathcal{V}$ eingeschränkt und dadurch natürlich auch die Menge der Modelle für eine TBox insgesamt.

[1] Wir erneuern nur die Definition für Subsumtion bezüglich *gfp*-Semantik, da nur dies Gegenstand der Untersuchung ist.

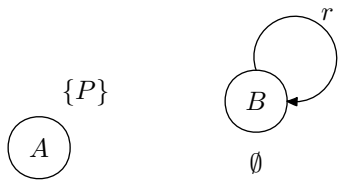

Abbildung 2.1: $\mathcal{EL}$-Beschreibungsgraph $\mathcal{G}_\mathcal{T}$ für $\mathcal{T}$ aus Beispiel 2.4.1

Wir wollen die Effekte von primitiven TBoxen betrachten; natürlich hat es Auswirkungen, wenn man zu einer TBox $\mathcal{T}$ noch eine primitive TBox $\mathcal{V}$ hinzufügt. So können durch primitive GCIs plötzlich ganz neue Subsumtionsbeziehungen zwischen den definierten Konzepten entstehen und das sowohl bezüglich deskriptiver Semantik als auch bezüglich *gfp*-Semantik.

Schauen wir auf ein kleines Beispiel:

Beispiel 2.4.1 Sei durch

$$\begin{array}{lcl} A & \equiv & P \\ B & \equiv & \exists r.B \\ & - - - & \\ P & \sqsubseteq & \exists r.P \end{array}$$

eine TBox $\mathcal{T}$ definiert mit primitiver TBox $\mathcal{V} = \{P \sqsubseteq \exists r.P\}$, $\mathbf{N}_{def} = \{A, B\}$, $\mathbf{N}_{prim} = \{P\}$ und $\mathbf{N}_r = \{r\}$. Der $\mathcal{EL}$-Beschreibungsgraph $\mathcal{G}_\mathcal{T}$ für die TBox $\mathcal{T}$ wird übrigens in Figur 2.1 gezeigt.

Das GCI in $\mathcal{V}$ bewirkt nun, dass $A^{\mathsf{gfp}(\mathcal{T},\mathcal{J})} \subseteq B^{\mathsf{gfp}(\mathcal{T},\mathcal{J})}$ für alle *gfp*-Modelle $\mathsf{gfp}(\mathcal{T},\mathcal{J})$ von $\mathcal{T}$ gilt, was ohne das primitive GCI nicht gelten würde. Die Begründung dafür lautet, dass für A auch gilt: $A \sqsubseteq_\mathcal{T} \exists r.A$ bezüglich deskriptiver Semantik. Also gilt auch bezüglich *gfp*-Semantik $A \sqsubseteq_{\mathcal{T},gfp} \exists r.A$. Da aber in jedem *gfp*-Modell $\mathsf{gfp}(\mathcal{T},\mathcal{J})$ für $\mathcal{T}$ für B gilt:

$$\begin{array}{rcl} B^{\mathsf{gfp}(\mathcal{T},\mathcal{J})} & = & \{\ x \mid x \in \Delta^\mathcal{J} \text{ und } \exists y \in \Delta^\mathcal{J} : (x,y) \in r^\mathcal{J}\} \\ & \supseteq & \{\ p_1 \mid p_1 \in P^\mathcal{J} \text{ und } \exists p_2 \in P^\mathcal{J} : (p_1,p_2) \in r^\mathcal{J}\} \\ & = & A^{\mathsf{gfp}(\mathcal{T},\mathcal{J})} \end{array}$$

wird A also von B bezüglich der hybriden TBox $(\mathcal{T} \cup \mathcal{V})$ mit *gfp*-Semantik subsumiert. Man sieht an diesem Beispiel auch, dass durch primitive GCIs ins-

besondere azyklische TBoxen zyklisch werden können. Das Verfahren zur Berechnung von Subsumtion bezüglich zyklischen $\mathcal{EL}$-TBoxen mit *gfp*-Semantik aus [3] würde diese — implizite — Subsumtionsbeziehung zwischen A und B jedoch nicht zeigen können; dass dort beschriebene Verfahren gilt nur für den Sonderfall, dass $\mathcal{V} = \emptyset$, wir zeigen aber im nächsten Kapitel, wie hybride TBoxen in normale (zyklische) TBoxen umgewandelt werden können unter Wahrung der *gfp*-Subsumtionsbeziehungen der Konzepte.

Kapitel 3

Reduktion hybrider $\mathcal{EL}$-TBoxen auf einfache $\mathcal{EL}$-TBoxen

Wir werden nun ein formales Verfahren angegeben, um Subsumtion bezüglich *gfp*-Semantik für $\mathcal{EL}$-TBoxen mit assoziierten primitiven $\mathcal{EL}$-TBoxen zu berechnen. Wie wir in Beispiel 2.4.1 gesehen haben, läßt der in [3] angegebene Ansatz nicht ohne weiteres verwenden. Wir müssen einen Weg finden, die in der primitiven TBox enthaltene Information in die normale TBox zu transportieren, ohne dadurch eine TBox zu erzeugen, bezüglich derer plötzlich ganz andere Subsumtionsbeziehungen gelten. Ein wenig formaler gesprochen geht es darum, eine Transformation f für TBoxen $\mathcal{T}$ mit primitiver TBox $\mathcal{V}$ zu finden mit:

$$f(\mathcal{T}, \mathcal{V}) \; = \; \mathcal{T}_f$$

so dass für zwei Konzepte $A,\ B \in \mathbf{N}_{def}^{\mathcal{T}}$ gilt:

$$A \sqsubseteq_{\mathcal{T},\mathcal{V},gfp} B \quad \Leftrightarrow \quad A \sqsubseteq_{\mathcal{T}_f,gfp} B$$

Zunächst brauchen wir jedoch auch hier eine Art Normalform für hybride $\mathcal{EL}$-TBoxen, die wir '*vervollständigt*' nennen. Von nun ab nehmen wir außerdem an, dass die zyklische $\mathcal{EL}$-TBoxen in Normalform für Graphendarstellung vorliegen und die primitiven generellen TBoxen in Normalform zur Subsumtionsmengenberechnung (Definitionen 2.2.8 und 2.3.2).

3.1 Vervollständigung von $\mathcal{EL}$-TBoxen

Um die in der primitiven TBox enthaltene Information in die normale TBox zu transportieren, muss die normale TBox zunächst um zusätzliche Definitionen erweitert werden. Es muss sichergestellt werden, dass für jedes primitive Konzept aus $\mathbf{N}_{prim}$ und jede primitive Existenzrestriktion, die in der primitiven TBox vorkommt, ein definiertes Konzept in $\mathbf{N}_{def}$ existiert. Wir werden an den entsprechenden Stellen in Abschnitt 3.2 auf diese Definition verweisen, um zu erläutern, warum wir die Vervollständigung brauchen.

Definition 3.1.1 Vervollständigung von $\mathcal{T}$ nach $\mathcal{V}$
Sei $\mathcal{T}$ eine TBox und mit den definierten Konzeptnamen $\mathbf{N}_{def}$ und den primitiven Konzeptnamen $\mathbf{N}_{prim}$ und $\mathcal{V}$ eine generelle primitive TBox. Sei weiterhin mit $\mathcal{J}$ eine feste primitive Interpretation gegeben für $\mathcal{T}$ und $\mathcal{V}$ und gelte $\mathcal{J} \models \mathcal{V}$. Die Vervollständigung von $\mathcal{T}$ nach $\mathcal{V}$ geschieht durch folgende Schritte:

1. Für jeden Konzeptnamen $P \in \mathbf{N}_{prim}$, der in einem GCI aus $\mathcal{V}$ oder in der Definition eines Konzeptes $A \in \mathbf{N}_{def}$ vorkommt und für die kein definiertes Konzept $A_P \in \mathbf{N}_{def}$ existiert mit $A_P \equiv P$ in $\mathcal{T}$, fügen wir eine neue Konzeptdefinition $A_P \equiv P$ zu $\mathcal{T}$ hinzu, wobei A_P ein neuer Name in $\mathbf{N}_{def}$ ist.

2. Für jede existenzielle Restriktion $\exists r.P$, die in einem GCI in $\mathcal{V}$ vorkommt und für die keine Konzeptdefinition $A \equiv \exists r.A_P$ mit $A_P \equiv P$ in $\mathcal{T}$ existiert, fügen wir eine neue Konzeptdefinition $A_{\exists r.P} \equiv \exists r.A_P$ zu $\mathcal{T}$ hinzu, wobei $A_{\exists r.P}$ ein neuer Name in $\mathbf{N}_{def}$ ist.

Bemerkung: $\mathbf{N}_{prim}$ wird durch Vervollständigung nicht verändert und keine existierende Definition in $\mathcal{T}$ erweitert. Es ist leicht zu sehen, dass die Vervollständigung einer TBox $\mathcal{T}$ in polynomieller Zeit abhängig von der Größe von $\mathcal{T}$ und $\mathcal{V}$ geleistet werden kann und die Größe von $\mathcal{T}$ nur linear ansteigt. Trotzdem müssen wir zeigen, dass sich durch die Erweiterung keine Subsumtionsbeziehung $A \sqsubseteq_{\mathcal{T} \cup \mathcal{V}} B$ für zwei Konzepte $A \in \mathbf{N}_{def}$ und $B \in \mathbf{N}_{def}$ verändert. Dafür formulieren wir ein Lemma:

LEMMA 3.1.1 Sei $\mathcal{T}$ eine TBox mit primitiver TBox $\mathcal{V}$ sowie den definierten Konzepten $\mathbf{N}_{def}^{\mathcal{T}} = \{A_1, \ldots, A_k\}$ und den primitiven Konzepten $\mathbf{N}_{prim}$. Mit $\mathcal{T}'$ sei die nach $\mathcal{V}$ vervollständigte TBox zu $\mathcal{T}$ gegeben mit den definierten Konzepten $\mathbf{N}_{def}^{\mathcal{T}'} = \{A_1, \ldots, A_k, A_{k+1}, \ldots, A_{k+h}\}$ und den primitiven Konzepten $\mathbf{N}_{prim}$. Für zwei Konzepte $A,\ B \in \mathbf{N}_{def}^{\mathcal{T}}$ gilt:

$$A \sqsubseteq_{\mathcal{T}} B \quad \Rightarrow \quad A \sqsubseteq_{\mathcal{T}'} B$$

■

BEWEIS 3.1.1 Sei durch $\mathcal{J}$ eine feste primitive Interpretation für $\mathcal{T}$ bzw. $\mathcal{T}'$ gegeben mit $\mathcal{J} \models \mathcal{V}$. Mit $Int_{\mathcal{T}}(\mathcal{J})$ ist der vollständige Verband der Interpretationen für $\mathcal{T}$ bezeichnet sowie mit $Int_{\mathcal{T}'}(\mathcal{J})$ der für $\mathcal{T}'$. Sei $\mathcal{I}'$ ein Modell aus $Int_{\mathcal{T}'}(\mathcal{J})$ für $\mathcal{T}'$ und $\mathcal{I}$ das Modell aus $Int_{\mathcal{T}}(\mathcal{J})$ für $\mathcal{T}$, für das gilt: $\forall i \in \{1, \ldots, k\} :\ A_i^{\mathcal{I}} = A_i^{\mathcal{I}'}$. Da $\mathcal{I}' \models \mathcal{T}'$ gilt, wissen wir $\mathcal{O}_{\mathcal{T}',\mathcal{J}}(\mathcal{I}') = \mathcal{I}'$, d.h. $\mathcal{I}'$ ist ein Fixpunkt von $\mathcal{O}_{\mathcal{T}',\mathcal{J}}$. Dann muss aber auch die Interpretation $\mathcal{I}$ ein Fixpunkt von $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ sein, da die Definitionen der Konzepte $\{A_1, \ldots, A_k\}$ in $\mathcal{T}$ und $\mathcal{T}'$ übereinstimmen und $Int_{\mathcal{T}}(\mathcal{J})$ und $Int_{\mathcal{T}'}(\mathcal{J})$ auf der selben primitiven Interpretation basieren. Daraus folgt für zwei Konzepte $A,\ B \in \{A_1, \ldots, A_k\}$:

$$A \not\sqsubseteq_{\mathcal{T}'} B \quad \Rightarrow \quad A \not\sqsubseteq_{\mathcal{T}} B$$

und daraus wiederum sofort das Lemma 3.1.1

q.e.d

Bemerkung: Das Lemma 3.1.1 bezieht sich *nur* auf Konzepte A und B, die schon in der nicht vervollständigten TBox $\mathcal{T}$ definiert sind.

3.2 Die Transformation

In der Definition der Transformation sowie später im Beweis benötigen wir die folgende Hilfsdefinition.

DEFINITION 3.2.1 Sei $\mathcal{T}$ eine TBox und $A \in \mathbf{N}_{def}$; mit $\mathsf{DEF}(\mathcal{T}, A) = \{P_1, \ldots, P_n, \exists r_1.B_1 \ldots \exists r_m.B_m\}$ sei die Menge aller Konzepte definiert, die auf der rechten Seite der Definition von A in $\mathcal{T}$ vorkommen.

□

Wenn $\mathcal{T}$ normalisiert ist, kommen in $\mathsf{DEF}(\mathcal{T}, A)$ also nur primitive Konzepte vor zusammen mit Existenzrestriktionen der Form $\exists r.B$ mit $B \in \mathbf{N}_{def}$.

DEFINITION 3.2.2 Sei $\mathcal{T}$ eine vervollständigte $\mathcal{EL}$-TBox mit primitiver $\mathcal{EL}$-TBox $\mathcal{V}$, den definierten Konzeptnamen $\mathbf{N}_{def}$ sowie den primitiven Konzeptnamen $\mathbf{N}_{prim}$. Die transformierte TBox $\mathcal{T}_f$ wird gebildet durch:

1. Berechne die Subsumtionsmengen $S_*(B)$ für $(\mathcal{T} \cup \mathcal{V})$ bezüglich deskriptiver Semantik für jedes Konzept $B \in (\mathbf{N}_{def} \cup \mathbf{N}_{prim})$
2. $f(\mathcal{T}, \mathcal{V}) = \mathcal{T}_f$ entsteht nun aus $\mathcal{T}$ auf folgende Weise.
   - $\mathcal{T}_f$ hat die gleichen Mengen $\mathbf{N}_{prim}$ und $\mathbf{N}_{def}$ wie $\mathcal{T}$
   - Sei $A \in \mathbf{N}_{def}$ und wie folgt definiert in $\mathcal{T}$:

     $$A \equiv P_1 \sqcap \ldots \sqcap P_n \sqcap \exists r_1.B_1 \sqcap \ldots \sqcap \exists r_m.B_m,$$

     die Definition des korrespondierenden Konzeptes A in $\mathcal{T}_f$ ist dann gegeben durch

     $$\begin{aligned} A \quad \equiv \quad & P_1 \sqcap \ldots \sqcap P_n \sqcap \exists r_1.B_1 \sqcap \ldots \sqcap \exists r_m.B_m \\ & \sqcap Q_1 \sqcap \ldots \sqcap Q_l \sqcap \exists r_{m+1}.B_{m+1} \sqcap \ldots \sqcap \exists r_{m+o}.B_{m+o} \end{aligned}$$

     wobei $Q_i \in \{Q_1, \ldots, Q_l\}$ zur Definition von A hinzugefügt wird, gdw. $Q_i \in S_*(A)$ und $(Q_i \notin \{P_1, \ldots, P_n\})$. Eine existenzielle Restriktion $\exists r_{m+i}.B_{m+i}$ wird zu der Definition von A in $\mathcal{T}_f$ hinzugefügt, gdw. ein Konzept $A_{\exists r_{m+i}.B_{m+i}}$ in der Subsumtionsmenge $S_*(A)$ existiert mit $(A_{\exists r_{m+i}.B_{m+i}} \equiv \exists r_{m+i}.B_{m+i}) \in \mathcal{T}$ und $(\exists r_{m+i}.B_{m+i}) \notin \mathsf{DEF}(\mathcal{T}, A)$

□

Die transformierte Definition von A enthält also zum einen zusätzlich alle primitiven Konzepte, von denen A subsumiert wird. Zum anderen sind in der Definition von A jetzt alle Existenzrestriktionen explizit gemacht, die durch ein GCI der Form $P' \sqsubseteq \exists r.P''$ impliziert werden. Dies ist der Fall, wenn $P' \in \{P_1, \ldots, P_n, Q_1, \ldots, Q_l\}$ und ein Konzept $A_{\exists r.P''}$ in der Subsumtionsmenge $S_*(A)$ von A existiert, das als $A_{\exists r.P''} \equiv \exists r.A_{P''}$ in $\mathcal{T}$ definiert ist und außerdem $(\exists r.A_{P''})$ noch nicht auf der rechten Seite der Definition von A vorkommt. Die Vervollständigung garantiert, dass zu jeder primitiven Existenzrestriktion $\exists r.P''$

sowohl ein Konzept $A_{P''} \in \mathbf{N}_{def}$ mit $A_{P''} \equiv P''$ in $\mathcal{T}$ existiert, als auch ein Konzept $A_{\exists r.P''} \in \mathbf{N}_{def}$ mit $A_{\exists r.P''} \equiv \exists r.A_{P''}$.

THEOREM 3.2.1 Die transformierte $\mathcal{EL}$-TBox $f(\mathcal{T}, \mathcal{V})$ für eine $\mathcal{EL}$-TBox $\mathcal{T}$ mit assoziierter primitiver TBox $\mathcal{V}$ kann in polynomieller Zeit abhängig von $||\mathcal{T} \cup \mathcal{V}||$ berechnet werden.

■

BEWEIS 3.2.1 Die Vervollständigung kann in polynomieller Zeit berechnet werden und ebenso die Subsumtionsmengen für $(\mathcal{T} \cup \mathcal{V})$. Für jedes Konzept $A \in \mathbf{N}_{def}$ kann in polynomieller Zeit festgestellt werden, ob ein primitives Konzept P mit $P \in S_*(A)$ schon in der $\mathsf{DEF}(\mathcal{T}, A)$ vorkommt, bzw. ob ein ein Konzept $A_{\exists r.P} \in S_*(A)$ existiert, mit $(A_{\exists r.P} \equiv \exists r.A_P) \in \mathcal{T}$, so dass $\exists r.A_P \notin \mathsf{DEF}(\mathcal{T}, A)$. Also kann auch die gesamte Transformation in polynomieller Laufzeit berechnet werden.

q.e.d

Das folgende Theorem garantiert uns, dass durch die Transformation $f(\mathcal{T}, \mathcal{V})$ für eine TBox $\mathcal{T}$ mit assoziierter primitiver genereller TBox $\mathcal{V}$ alle *gfp*-Subsumtionsbeziehungen erhalten bleiben und auch keine neuen hinzugefügt werden.

THEOREM 3.2.2 Sei $\mathcal{T}$ eine $\mathcal{EL}$-TBox mit primitiver $\mathcal{EL}$-TBox $\mathcal{V}$ und $\mathcal{T}_f = f(\mathcal{T}, \mathcal{V})$ die transformierte TBox zu $\mathcal{T}$ und $\mathcal{V}$ — für zwei Konzepte A und B aus $\mathbf{N}_{def}$ gilt:

$$A \sqsubseteq_{\mathcal{T}_f, gfp} B \quad \Leftrightarrow \quad A \sqsubseteq_{\mathcal{T}, \mathcal{V}, gfp} B$$

■

Wir werden im Beweis dieses Theorems noch eine Hilfsdefinitionen brauchen:

DEFINITION 3.2.3 Sei $\mathcal{T}$ eine TBox, $\mathcal{V}$ eine primitive TBox und $\mathcal{T}_f = f(\mathcal{T}, \mathcal{V})$ die transformierte TBox, sowie $A \in \mathbf{N}_{def}$; mit $\mathsf{oterm}(A)$ bezeichnen wir die rechte Seite der Definition von A in der originalen TBox $\mathcal{T}$ und mit $\mathsf{fterm}(A)$ den Teil der Konzeptdefinition von A in der transformierten TBox $\mathcal{T}_f$, der durch die Transformation hinzugefügt wurde, d.h. in der originalen TBox $\mathcal{T}$ ist A definiert als $A \equiv \mathsf{oterm}(A)$ und in der transformierten TBox $\mathcal{T}_f$ als $A \equiv \mathsf{oterm}(A) \sqcap \mathsf{fterm}(A)$.

□

Beweis 3.2.2 von Theorem 3.2.2

Zunächst soll die allgemeine Beweisidee vorgestellt werden. Wir zeigen als erstes, dass

$$A \sqsubseteq_{\mathcal{T},\mathcal{V},gfp} B \quad \Leftrightarrow \quad A \sqsubseteq_{\mathcal{T}_f,\mathcal{V},gfp} B, \tag{i}$$

also dass alle *gfp*-Subsumtionsbeziehungen, die zwischen Konzepten A und B bezüglich einer TBox $\mathcal{T}$ mit primitiver TBox $\mathcal{V}$ gelten, auch für die transformierte TBox $\mathcal{T}_f$ mit primitiver TBox $\mathcal{V}$ gelten und umgekehrt. Als zweites zeigen wir

$$A \sqsubseteq_{\mathcal{T}_f,\mathcal{V},gfp} B \quad \Leftrightarrow \quad A \sqsubseteq_{\mathcal{T}_f,gfp} B, \tag{ii}$$

also dass die primitiven GCIs in der transformierten TBox redundant sind. Aus (i) und (ii) folgt dann direkt die Gültigkeit von Theorem 3.2.2

Teil I:

Sei $\mathcal{T}$ eine TBox mit $\mathbf{N}_{def} = \{A_1, \ldots A_k\}$ und primitiver TBox $\mathcal{V}$ und sei $\mathcal{T}_f$ die transformierte TBox zu $\mathcal{T}$. Mit $\mathcal{J}$ sei eine beliebige primitive Interpretation für $\mathcal{T}$ bzw. $\mathcal{T}_f$ gegeben mit $\mathcal{J} \models \mathcal{V}$ und durch $Int(\mathcal{J})$ der gemeinsame vollständige Verband der Interpretationen für $\mathcal{T}$ bzw. $\mathcal{T}_f$, die auf $\mathcal{J}$ basieren. Um zu beweisen, dass $A \sqsubseteq_{\mathcal{T},\mathcal{V},gfp} B$ gilt, gdw. auch $A \sqsubseteq_{\mathcal{T}_f,\mathcal{V},gfp} B$ gilt, reicht es aus, zu zeigen, dass das *gfp*-Modell $\mathsf{gfp}(\mathcal{T},\mathcal{J}) \in Int(\mathcal{J})$ der ursprünglichen TBox auch das *gfp*-Modell der transformierten TBox $\mathcal{T}_f$ ist. Dafür zeigen wir zunächst, dass jedes Modell von $\mathcal{T}$ auch ein Modell von $\mathcal{T}_f$ ist und folglich auch $\mathsf{gfp}(\mathcal{T},\mathcal{J}) \models \mathcal{T}_f$ gilt. Anschließend zeigen wir, dass für $\mathcal{T}_f$ kein größeres Modell in $Int(\mathcal{J})$ existiert als das *gfp*-Modell der ursprünglichen TBox $\mathcal{T}$ — in Zeichen[1]

$$\forall\, \mathcal{I}_f \in Int(\mathcal{J}) : (\mathcal{I}_f \models \mathcal{T}_f \;\Rightarrow\; \mathcal{I}_f \preceq_{\mathcal{J}} \mathsf{gfp}(\mathcal{T},\mathcal{J}))\,.$$

Zunächst zeigen wir, dass jedes Modell $\mathcal{I} \in Int(\mathcal{J})$ von $\mathcal{T}$ auch ein Modell von $\mathcal{T}_f$ ist. Sei $A \in \mathbf{N}_{def}$ ein beliebiges Konzept, welches in $\mathcal{T}$ bzw. $\mathcal{T}_f$ definiert ist. Da $\mathcal{J} \models \mathcal{V}$ und $\mathcal{I} \in Int(\mathcal{J})$ gelten auch die deskriptiven Konsequenzen der GCIs

[1] Mit $\preceq_{\mathcal{J}}$ ist eine Halbordnung auf $Int(\mathcal{J})$ definiert und die Modelle $\mathsf{gfp}(\mathcal{T},\mathcal{J})$ und $\mathcal{I}_f$ müssen nicht unbedingt über diese Halbordnungsrelation vergleichbar sein - nur würde, wenn sie es nicht wären, noch ein weiteres, größeres Modell $\mathcal{I}'_f$ von $\mathcal{T}_f$ existieren, mit $\mathcal{I}_f \prec \mathcal{I}'_f$ und $\mathsf{gfp}(\mathcal{T},\mathcal{J}) \prec \mathcal{I}'_f$. Daher muss $\mathsf{gfp}(\mathcal{T},\mathcal{J})$ auch ein *gfp*-Modell für $\mathcal{T}_f$ sein, wenn kein Modell $\mathcal{I}_f$ von $\mathcal{T}_f$ größer als $\mathsf{gfp}(\mathcal{T},\mathcal{J})$ ist und $\mathsf{gfp}(\mathcal{T},\mathcal{J}) \models \mathcal{T}_f$.

aus $\mathcal{V}$ für alle $A \in \mathbf{N}_{def}$; d.h. gdw. ein Konzept F aus $\mathbf{N}_{def} \cup \mathbf{N}_{prim}$ Element der Subsumtionsmenge $S_*(A)$ ist, gilt auch $A \sqsubseteq_{\mathcal{T} \cup \mathcal{V}} F$ in allen Modellen $\mathcal{I} \in Int(\mathcal{J})$. Daher gilt auch: $A^{\mathcal{I}} \subseteq \mathsf{fterm}(A)^{\mathcal{I}}$ und weil gleichzeitig $A^{\mathcal{I}} = \mathsf{oterm}(A)^{\mathcal{I}}$ gilt, folgt auch $A^{\mathcal{I}} = (\mathsf{oterm}(A) \sqcap \mathsf{fterm}(A))^{\mathcal{I}}$ für alle $\mathcal{I} \in Int(\mathcal{J})$ mit $\mathcal{I} \models \mathcal{T}$. Das bedeutet aber $\mathcal{I} \models \mathcal{T}_f$. Damit haben wir gezeigt, dass jedes Modell von $\mathcal{T}$ auch ein Modell von $\mathcal{T}_f$ ist.

Es bleibt noch übrig zu zeigen, dass das *gfp*-Modell $\mathsf{gfp}(\mathcal{T}, \mathcal{J})$ auch für die transformierte TBox $\mathcal{T}_f$ ein *gfp*-Modell ist. Die *gfp*-Modelle $\mathsf{gfp}(\mathcal{T}, \mathcal{J})$ bzw. $\mathsf{gfp}(\mathcal{T}_f, \mathcal{J})$ werden durch iterative Anwendung der Fixpunktoperatoren $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ bzw. $\mathcal{O}_{\mathcal{T}_f,\mathcal{J}}$ gefunden. Da weder $\mathcal{O}_{\mathcal{T}_f,\mathcal{J}}$ noch $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ mit Notwendigkeit *abwärts stetig* sind, muss nach einer einfachen ω-Iteration die Interpretation $\mathcal{I}^{\downarrow\omega} = glb(\mathcal{I}^{\downarrow n} \mid n \in \mathbb{N})$ kein Fixpunkt von $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ bzw. $\mathcal{O}_{\mathcal{T}_f,\mathcal{J}}$ sein. Tarskis Fixpunkttheorem [23] sagt aber, dass eine Ordinalzahl α existiert, mit $\mathcal{I}^{\downarrow\alpha}$ der größte Fixpunkt von $\mathcal{O}_{\mathcal{T},\mathcal{J}}$ ist. Wir zeigen durch transfinite Induktion über β für die Fixpunktoperatoren $\mathcal{O}_{\mathcal{T}_f,\mathcal{J}}$ bzw. $\mathcal{O}_{\mathcal{T},\mathcal{J}}$, dass für alle $\beta \leq \alpha$ gilt $\mathcal{I}_f^{\downarrow\beta} \preceq_{\mathcal{J}} \mathcal{I}^{\downarrow\beta}$ mit $\mathcal{I}_f^{\downarrow\beta} = \mathcal{O}^{\beta}_{\mathcal{T}_f,\mathcal{J}}(\mathcal{I}_{top})$ bzw. $\mathcal{I}^{\downarrow\beta} = \mathcal{O}^{\beta}_{\mathcal{T},\mathcal{J}}(\mathcal{I}_{top})$. Hierbei ist $\mathcal{O}^{\beta}_{\mathcal{T},\mathcal{J}}(\mathcal{I}_{top})$ eine Kurzschreibweise für $\underbrace{\mathcal{O}_{\mathcal{T},\mathcal{J}}(\ldots(\mathcal{O}_{\mathcal{T},\mathcal{J}}}_{\beta \text{ mal}}(\mathcal{I}_{top})\ldots)$, der β-fachen Applikation von $\mathcal{O}_{\mathcal{T},\mathcal{J}}$.

Induktionsbasis $\beta := 0$ Dann gilt offensichtlich $A_i^{\mathcal{I}_f^{\downarrow\beta}} = A_i^{\mathcal{I}_{top}} = A_i^{\mathcal{I}^{\downarrow\beta}}$ und also auch $\mathcal{I}_f^{\downarrow\beta} \preceq_{\mathcal{J}} \mathcal{I}^{\downarrow\beta}$ für alle $\forall A_i \in \mathbf{N}_{def}$.

Induktionsschritt Hier müssen wir zwei Fälle unterscheiden, einmal den Fall, dass α eine gewöhnliche Nachfolger-Ordinalzahl ist, und zum anderen, dass α eine Grenzwert-Ordinalzahl ist.

Fall 1 α ist Nachfolger-Ordinalzahl und gelte für $\beta < \alpha$ die Behauptung, d.h. wir nehmen an, dass für $\mathcal{I}_f^{\downarrow\beta} = \mathcal{O}^{\beta}_{\mathcal{T}_f,\mathcal{J}}(\mathcal{I}_{top})$ bzw. $\mathcal{I}^{\downarrow\beta} = \mathcal{O}^{\beta}_{\mathcal{T},\mathcal{J}}(\mathcal{I}_{top})$ gilt: $\mathcal{I}_f^{\downarrow\beta} \preceq_{\mathcal{J}} \mathcal{I}^{\downarrow\beta}$ d.h. dass für alle A_i mit $1 \leq i \leq k$ und $(A_i \equiv P_{i,1} \sqcap \ldots \sqcap P_{i,n} \sqcap \exists r_{i,1}.B_{i,1} \sqcap \ldots \sqcap \exists r_{i,m}.B_{i,m}) \in \mathcal{T}$ bzw. $(A_i \equiv \mathsf{oterm}(A_i) \sqcap Q_{i,1} \sqcap \ldots \sqcap Q_{i,l} \sqcap \exists r_{i,m+1}.B_{i,m+1} \sqcap \ldots \sqcap \exists r_{i,m+o}.B_{i,m+o}) \in \mathcal{T}_f$ gilt:

$$A_i^{\mathcal{I}_f^{\downarrow\beta}} \subseteq A_i{}^{\mathcal{I}^{\downarrow\beta}}.$$

Wenn wir nun die Fixpunktoperatoren ein weiteres mal applizieren,

so erhalten wir $\mathcal{I}^{\downarrow\beta+1}$ bzw. $\mathcal{I}_f^{\downarrow\beta+1}$ mit

$$A_i^{\mathcal{I}^{\downarrow\beta+1}} = \left(\bigsqcap_{j_1 \leq n} P_{i,j_1} \sqcap \bigsqcap_{j_2 \leq m} (\exists r_{i,j_2}.B_{i,j_2}) \right)^{\mathcal{I}^{\downarrow\beta}},$$

bzw. da die Interpretation für die primitiven Konzepte P_{i,j_1} unveränderlich ist

$$A_i^{\mathcal{I}^{\downarrow\beta+1}} = \bigcap_{j_1 \leq n} P_{i,j_1}^{\mathcal{J}} \cap \left(\bigsqcap_{j_2 \leq m} (\exists r_{i,j_2}.B_{i,j_2}) \right)^{\mathcal{I}^{\downarrow\beta}}. \tag{iii}$$

Für die transformierte TBox gilt analog:

$$\begin{aligned} A_i^{\mathcal{I}_f^{\downarrow\beta+1}} &= \bigcap_{j_1 \leq n} P_{i,j_1}^{\mathcal{J}} \cap \left(\bigsqcap_{j_2 \leq m} (\exists r_{i,j_2}.B_{i,j_2}) \right)^{\mathcal{I}_f^{\downarrow\beta}} \\ &\cap \bigcap_{j_3 \leq l} Q_{i,j_3}^{\mathcal{J}} \cap \left(\bigsqcap_{m+1 \leq j_4 \leq m+o} (\exists r_{i,j_4}.B_{i,j_4}) \right)^{\mathcal{I}_f^{\downarrow\beta}}. \end{aligned} \tag{iv}$$

Aus den Gleichungen (iii) und (iv) wird deutlich, dass $\mathcal{I}_f^{\downarrow\beta+1} \prec \mathcal{I}^{\downarrow\beta+1}$ bewiesen ist, wenn wir zeigen können, dass

$$\left(\bigsqcap_{j_2 \leq m} (\exists r_{i,j_2}.B_{i,j_2}) \right)^{\mathcal{I}_f^{\downarrow\beta}} \subseteq \left(\bigsqcap_{j_2 \leq m} (\exists r_{i,j_2}.B_{i,j_2}) \right)^{\mathcal{I}^{\downarrow\beta}}.$$

Die obige Subsumtionsbeziehung gilt aber, wenn

$$\forall j_2 \in \{1 \ldots m\} : (\exists r_{i,j_2}.B_{i,j_2})^{\mathcal{I}_f^{\downarrow\beta}} \subseteq (\exists r_{i,j_2}.B_{i,j_2})^{\mathcal{I}^{\downarrow\beta}}.$$

Fixieren also ein beliebiges j_2. Wir wissen per Induktionsvoraussetzung, dass $(B_{i,j_2})^{\mathcal{I}_f^{\downarrow\beta}} \subseteq (B_{i,j_2})^{\mathcal{I}^{\downarrow\beta}}$, und weil

$$(\exists r_{i,j_2}.B_{i,j_2})^{\mathcal{I}_f^{\downarrow\beta}} = \{x \mid \exists\ (x,y) \in r_{i,j_2}^{\mathcal{J}} \text{ mit } y \in (B_{i,j_2})^{\mathcal{I}_f^{\downarrow\beta}}\}$$

bzw.

$$(\exists r_{i,j_2}.B_{i,j_2})^{\mathcal{I}^{\downarrow\beta}} = \{x \mid \exists\ (x,y) \in r_{i,j_2}^{\mathcal{J}} \text{ mit } y \in (B_{i,j_2})^{\mathcal{I}^{\downarrow\beta}}\}$$

gilt offensichtlich auch

$$(\exists r_{i,j_2}.B_{i,j_2})^{\mathcal{I}_f^{\downarrow\beta}} \subseteq (\exists r_{i,j_2}.B_{i,j_2})^{\mathcal{I}^{\downarrow\beta}}.$$

Fall 2 Sei α eine Grenzwert-Ordinalzahl, gelte für $\beta < \alpha$ die Behauptung; dann gilt

$$A^{\mathcal{I}_f^{\downarrow\alpha}} = A^{glb(\mathcal{I}_f^{\downarrow\beta})} = \bigcap_{\beta<\alpha} A^{\mathcal{I}_f^{\downarrow\beta}} \subseteq \bigcap_{\beta<\alpha} A^{\mathcal{I}^{\downarrow\beta}} = A^{glb(\mathcal{I}^{\downarrow\beta})}.$$

Da für jeden Iterationsschritt β gilt $\mathcal{I}_f^{\downarrow\beta} \preceq_{\mathcal{J}} \mathcal{I}^{\downarrow\beta}$, gilt dies auch in dem Iterationsschritt α, mit dem $\mathcal{I}^{\downarrow\alpha} = \mathsf{gfp}(\mathcal{T}, \mathcal{J})$ - also selbst wenn wir das *gfp*-Modell $\mathsf{gfp}(\mathcal{T}_f, \mathcal{J})$ in einem früheren Iterationsschritt α_f finden, sichert uns obiger Induktionsbeweis zu, dass $\mathsf{gfp}(\mathcal{T}_f, \mathcal{J}) \preceq_{\mathcal{J}} \mathsf{gfp}(\mathcal{T}, \mathcal{J})$, woraus aber mit der Tatsache, dass jedes Modell von $\mathcal{T}$ in $Int(\mathcal{J})$ auch ein Modell von $\mathcal{T}_f$ ist, $\mathsf{gfp}(\mathcal{T}, \mathcal{J}) = \mathsf{gfp}(\mathcal{T}_f, \mathcal{J})$ folgt.

Teil II:

Wir zeigen, dass $A \sqsubseteq_{\mathcal{T}_f,\mathcal{V},gfp} B$ gilt, gdw. auch $A \sqsubseteq_{\mathcal{T}_f,gfp} B$ gilt, wobei die Richtung ($\Leftarrow$) trivialerweise gilt.

($\Rightarrow$) Wir zeigen, dass die folgende Implikation gültig ist:

$$A \not\sqsubseteq_{\mathcal{T}_f,gfp} B \;\Rightarrow\; A \not\sqsubseteq_{\mathcal{T}_f,\mathcal{V},gfp} B, \tag{v}$$

daraus folgt direkt, dass mit $A \sqsubseteq_{\mathcal{T}_f,\mathcal{V},gfp} B$ auch $A \sqsubseteq_{\mathcal{T}_f,gfp} B$ gilt.

Voraussetzung ist also $A \not\sqsubseteq_{\mathcal{T}_f,gfp} B$. Um die Gültigkeit von (v) zu beweisen, reicht es aus, zu zeigen, dass sich stets eine primitive Interpretation $\mathcal{J}$ finden lässt mit $\mathcal{J} \models \mathcal{V}$, so dass im *gfp*-Modell $\mathsf{gfp}(\mathcal{T}_f, \mathcal{J})$ gilt:

$$A^{\mathsf{gfp}(\mathcal{T}_f,\mathcal{J})} \not\subseteq B^{\mathsf{gfp}(\mathcal{T}_f,\mathcal{J})} \tag{vi}$$

Sei $\mathcal{G}_{\mathcal{T}_f} = (\mathbf{N}_{def}, E_{\mathcal{T}_f}, L_{\mathcal{T}_f})$ der $\mathcal{EL}$-Beschreibungsgraph zu $\mathcal{T}_f$. Wir konstruieren die kanonische primitive Interpretation $\mathcal{K}$ aus $\mathcal{G}_{T_f}$ wie folgt:

- Für jedes $A_i \in \mathbf{N}_{def}$ existiert genau ein $a_i \in \Delta^{\mathcal{K}}$.
- $a_i \in P^{\mathcal{K}} \Leftrightarrow P \in L_{\mathcal{T}_f}(A_i) \Leftrightarrow P \in \mathsf{DEF}(\mathcal{T}_f, A_i)$
- $(a_1, a_2) \in r^{\mathcal{K}} \Leftrightarrow (A_1, r, A_2) \in E_{\mathcal{T}_f} \Leftrightarrow \exists r.A_2 \in \mathsf{DEF}(\mathcal{T}_f, A_1)$.

Bemerkung 1: Die Kleinbuchstaben für die definierten Konzepte führen wir ein, um im folgenden besser unterscheiden zu können, ob wir von einem Konzept

aus $\mathbf{N}_{def}$ oder einem Element aus $\Delta^{\mathcal{K}}$ sprechen. Durch die Konstruktion von $\mathcal{K}$ ist eine bijektive Abbildung $\varphi_{\mathcal{T}_f} : \mathbf{N}_{def} \rightarrow \Delta^{\mathcal{K}}$ induziert mit $\varphi_{\mathcal{T}_f}(A_i) = a_i$.

Sei weiterhin mit $\Phi_{\mathcal{T}_f} : \mathbf{N}_{def} \times \Delta^{\mathcal{K}}$ eine binäre Relation definiert mit

$$(A_i, a_i) \in \Phi_{\mathcal{T}_f} \;\Leftrightarrow\; \varphi_{\mathcal{T}_f}(A_i) = a_i$$

Bemerkung 2: Wenn $\mathcal{G}_{\mathcal{K}} := (\Delta^{\mathcal{K}}, E_{\mathcal{K}}, L_{\mathcal{K}})$ der Beschreibungsgraph der primitiven Interpretation $\mathcal{K}$ ist, gilt also $\Phi_{\mathcal{T}_f}$ ist eine Simulation von $\mathcal{G}_{\mathcal{T}_f}$ nach $\mathcal{G}_{\mathcal{K}}$ bzw. $\Phi^{-1}_{\mathcal{T}_f}$ ist eine Simulation von $\mathcal{G}_{\mathcal{K}}$ nach $\mathcal{G}_{\mathcal{T}_f}$.

Betrachten wir nun das *gfp*-Modell $\mathsf{gfp}(\mathcal{T}_f, \mathcal{K})$ — mit Proposition 18 aus [3] gilt: Wenn $x \in A^{\mathsf{gfp}(\mathcal{T}_f, \mathcal{K})}$ ist, genau dann existiert eine Simulation von $Z : \mathcal{G}_{\mathcal{T}_f} \rightleftharpoons \mathcal{G}_{\mathcal{K}}$ so dass $(A, x) \in Z$. Daraus folgt direkt mit den obigen Überlegungen

$$\forall a \in \Delta^{\mathcal{K}}, A \in \mathbf{N}_{def} : \; (A, a) \in \Phi_{\mathcal{T}_f} \;\Rightarrow\; a \in A^{\mathsf{gfp}(\mathcal{T}_f, \mathcal{K})}$$

Weiterhin gilt $a \notin B^{\mathsf{gfp}(\mathcal{T}_f, \mathcal{K})}$, sonst gäbe es eine Simulation $Y : \mathcal{G}_{\mathcal{T}_f} \rightleftharpoons \mathcal{G}_{\mathcal{K}}$ mit $(B, a) \in Y$. Dann aber würde für die Simulation $Z : \mathcal{G}_{\mathcal{T}_f} \rightleftharpoons \mathcal{G}_{\mathcal{T}_f}$ mit $Z := Y \circ \Phi^{-1}_{\mathcal{T}_f}$ gelten, dass $(B, A) \in Z$. Dann aber würde mit Theorem 19 aus [3] gelten $A \sqsubseteq_{\mathcal{T}_f, gfp} B$, und dies wäre ein Widerspruch zu unserer Voraussetzung.

Es bleibt übrig zu zeigen, dass $\mathcal{K} \models \mathcal{V}$:

1. Sei $P \sqsubseteq Q$ eine deskriptive Konsequenz aus $\mathcal{V}$ — das deckt trivialerweise auch den Fall ab, dass $P \sqsubseteq Q$ ein GCI aus $\mathcal{V}$ ist — und gelte für ein Element $a \in \Delta^{\mathcal{K}}$, dass $a \in P^{\mathcal{K}}$. Dann gilt $P \in L_{\mathcal{K}}(a))$ und also $P \in L_{\mathcal{T}_f}(A)$ und also $P \in \mathsf{DEF}(\mathcal{T}, A)$, wobei $\Phi_{\mathcal{T}_f}(A) = a$ gelte. Durch Konstruktion der transformierten TBox $\mathcal{T}_f$ folgt aber $Q \in \mathsf{DEF}(\mathcal{T}_f, A)$. Also gilt $Q \in L_{\mathcal{T}_f}(A)$ und also $a \in Q^{\mathcal{K}}$.

2. Sei $P_1 \sqcap P_2 \sqsubseteq Q$ ein GCI in $\mathcal{V}$ und $a \in \Delta^{\mathcal{K}}$ ein Element mit $a \in P_1^{\mathcal{K}}$ und $a \in P_2^{\mathcal{K}}$. Dann enthalten $L_{\mathcal{K}}(a)$, $L_{\mathcal{T}_f}(A)$ und $\mathsf{DEF}(\mathcal{T}_f, A)$ die Konzepte P_1 und P_2. Durch Konstruktion von $\mathcal{T}_f$ enthält $\mathsf{DEF}(\mathcal{T}_f, A)$ auch Q und $Q \in L_{\mathcal{T}_f}(A)$ und $Q \in L_{\mathcal{K}}(a)$. Daraus folgt $a \in Q^{\mathcal{K}}$.

3. Sei $P \sqsubseteq \exists Q.$ ein GCI in $\mathcal{V}$ und $a \in \Delta^{\mathcal{K}}$ ein Element mit $a \in P_1^{\mathcal{K}}$. Dann enthalten $L_{\mathcal{K}}(a)$, $L_{\mathcal{T}_f}(A)$ und $\mathsf{DEF}(\mathcal{T}_f, A)$ das Konzepte P Also enthält $\mathsf{DEF}(\mathcal{T}_f, A)$ auch ein Konzept $\exists r.A_Q$ mit $Q \in L_{\mathcal{T}_f}(A_Q)$ und $Q \in L_{\mathcal{K}}(a_q)$

mit $\Phi_{\mathcal{T}_f}(A_Q) = a_q$. Durch Konstruktion von $\mathcal{T}_f$ gilt $\exists r.A_Q \in DEF(\mathcal{T}_f, A)$ und es gibt also in $E_{\mathcal{T}_f}$ eine Kante (A, r, A_Q) und in $E_{\mathcal{K}}$ eine Kante $(\Phi_{\mathcal{T}_f}(A), r, \Phi_{\mathcal{T}_f}(A_Q))$ und also gilt $(a, a_q) \in r^{\mathcal{K}}$ und $a_q \in Q^{\mathcal{K}}$; das bedeutet $a \in (\exists r.Q)^{\mathcal{K}}$.

4. Sei $\exists r.P \sqsubseteq Q$ ein GCI in $\mathcal{V}$ und $a_1 \in \Delta^{\mathcal{K}}$ ein Element, für dass ein $a_2 \in \Delta^{\mathcal{K}}$ existiert mit $a_2 \in P^{\mathcal{K}}$ und $(a_1, a_2) \in r^{\mathcal{K}}$. Dann gilt also $P \in L_{\mathcal{K}}(a_2)$, $P \in L_{\mathcal{T}_f}(A_2)$ und $P \in \mathsf{DEF}(\mathcal{T}_f, A_2)$. Weiterhin gilt, es existiert in $E_{\mathcal{K}}$ eine Kante (a_1, r, a_2) und in $E_{\mathcal{T}_f}$ eine Kante (A_1, r, A_2) und also $\exists r.A_2 \in \mathsf{DEF}(\mathcal{T}_f, A_1)$. Durch Konstruktion von $\mathcal{T}_f$ gilt also auch $Q \in \mathsf{DEF}(\mathcal{T}_f, A_1)$ und also $Q \in L_{\mathcal{T}_f}(A_1)$ und $Q \in L_{\mathcal{K}}(a_1)$. Also $a_1 \in Q^{\mathcal{K}}$.

Da in normalisierten primitiven TBoxen $\mathcal{V}$ nur die oben behandelten vier Arten von GCIs auftauchen können, wissen wir, dass durch $\mathcal{K}$ keine deskriptives Konsequenz aus $\mathcal{V}$ verletzt wird und damit $\mathcal{K} \models \mathcal{V}$. Da wir also gezeigt haben, wie sich unter der Voraussetzung, dass $A \not\sqsubseteq_{\mathcal{T}_f, gfp} B$ stets eine primitive Interpretation $\mathcal{J}$ finden lässt mit $\mathcal{J} \models \mathcal{V}$, so dass außerdem für das *gfp*-Modell $\mathsf{gfp}(\mathcal{T}_f, \mathcal{J})$ auch (vi) gilt, können wir auch $A \not\sqsubseteq_{\mathcal{T}_f, \mathcal{V}, gfp} B$ folgern, was den Beweis für (ii) vollendet.

Aus (i) und (ii) folgt unmittelbar Theorem 3.2.2.

q.e.d

Aus Theorem 3.2.1 und 3.2.2 sowie den Ergebnissen aus [9] und [3] folgt auch

THEOREM 3.2.3 Das Subsumtionsproblem für hybride $\mathcal{EL}$-TBoxen ist in polynomieller Zeit entscheidbar.

■

Kapitel 4

Der Hybrid-$\mathcal{EL}$-Reasoner

Da das Verfahren zur *gfp*-Subsumtionsberechnung für hybride $\mathcal{EL}$-TBoxen erst in der vorliegenden Arbeit entwickelt wurde, gibt es soweit uns bekannt ist keine geeigneten realen Wissensbasen, anhand derer die Implementation getestet werden könnte. Die prototypische Implementierung ist auch noch nicht in einem Stadium, in dem etwa *benchmark*-Tests sinnvoll wären. Wir dokumentieren daher den Verlauf und die Ergebnisse zweier Funktionstests, um zu zeigen, dass die Implementierung im Prinzip funktioniert. Zuvor jedoch noch eine Installationsanleitung und einige allgemeine Anmerkungen zur Bedienung. Für eine Dokumentation für Programmierer sei auf den Anhang verwiesen.

4.1 Allgemeines

Um die Nachvollziehbarkeit der Tests zu gewährleisten, beschreiben wir kurz, unter welchen Bedingungen der Hybrid-$\mathcal{EL}$-Reasoner entwickelt und getestet wurde.

Architektur: PC mit Intel™ x86-Architektur mit SuSE™ Linux 9.0 (i586), VERSION = 9.0 Betriebssystem

LISP Interpreter: CMU Common Lisp, VERSION=18e. [18]

IDE: Emacs 21.3.1

Der Hybrid-$\mathcal{EL}$-Reasoner umfasst neben der Datei `hybrid-reasoner.lisp` die Dateien `gfp.lisp` und `gfp.x86f`, in welcher der *gfp*-Subsumtionsalgorithmus für zyklische $\mathcal{EL}$-TBoxen implementiert ist, sowie die Datei `gci.lisp` und `gci.x86f`[1], die den Subsumtionsalgorithmus für generelle $\mathcal{EL}$-TBoxen implementiert. Beide Programme wurden von B. Suntisrivaraporn an der Technischen Universität Dresden entwickelt [22]. Der Hybrid-$\mathcal{EL}$-Reasoner benutzt diese beiden Implementationen; wegen Namensraumkollisionen mussten jedoch beide Dateien leicht modifiziert werden und es ist also nicht möglich, die Dateien einfach durch neuere Versionen zu ersetzen. Die Installation erfordert als Minimumvoraussetzung eine LISP-Umgebung, aber es wurden bisher noch keine Tests in einer anderen Umgebung als der oben beschriebenen durchgeführt. Es können daher bislang auch keine Aussagen darüber gemacht werden, wie der Hybrid-$\mathcal{EL}$-Reasoner sich unter anderen Bedingen verhält.

4.1.1 Installation und Bedienung

Nach der Installation der Dateien `hybrid-reasoner.lisp`, `gfp.lisp` und `gci.lisp` in ein beliebiges Verzeichnis startet man in einer ‚*shell*'einen LISP-Interpreter. Dann wird der Hybrid-$\mathcal{EL}$-Reasoner mit dem Befehl

```
* (load "hybrid-reasoner.lisp")
```

gestartet. Wenn man die $\mathcal{EL}$-TBox in einer Datei `tbox.lisp` und die GCIs in einer Datei `vbox.lisp` im gleichen Verzeichnis gespeichert hat, startet man mit dem Befehl

```
* (run "tbox.lisp" "vbox.lisp")
```

die Transformation. Dadurch wird zunächst eine vervollständigte hybride $\mathcal{EL}$-TBox erzeugt, die in der Datei `completed-tbox.lisp` gespeichert wird. Die vervollständigte hybride $\mathcal{EL}$-TBox wird daraufhin transformiert und in einer Datei `transformed-tbox.lisp` gespeichert. Abschließend wird der $\mathcal{EL}$-Reasoner `gfp.lisp` mit dieser Datei gestartet und die *gfp*-Klassifikation, d.h. alle Subsumtionsbeziehungen für die transformierte TBox mit *gfp*-Semantik berechnet.

[1] Die Dateiendung `.x86f` bedeutet, dass die Datei *byte compiliert* wurde — dies bewirkt i.d.R. eine signifikante Laufzeitoptimierung gegenüber den uncompilierten Versionen.

Bemerkung: Es empfiehlt sich, zumindestens alle externen Programme[2] in einer byte-compilierten Version zu benutzen, was der Voreinstellung entspricht. Es mag aber gute Gründe geben, trotzdem die uncompilierten Programme zu nutzen, z.B. beim Integrieren neuerer Versionen davon. Dann müssen in der Datei `hybrid-reasoner.lisp` zuvor die beiden Zeilen

```
* (load "gci.x86f")
* (load "gfp.x86f")
```

zu

```
* (load "gci.lisp")
* (load "gfp.lisp")
```

geändert werden.

4.1.2 Formatierungsvorschriften für die TBoxen:

Die Dateien `tbox.lisp` und `vbox.lisp` müssen den folgenden Standards entsprechen.

Konzeptdefinitionen in zyklischen und/oder generellen $\mathcal{EL}$-TBoxen haben das Format

```
(DEFCONCEPT A1 (AND P1 P2 (SOME R B)))
(DEFCONCEPT B (SOME R B))
(DEFCONCEPT A2 P1)
```

GCIs haben das Format

```
(IMPLIES  C D)
```

wobei `C` und `D` komplexe $\mathcal{EL}$-Konzeptbeschreibungen sein können, also Terme wie z.B. `(AND P1 P2 (SOME R Q))`. Dabei ist es wichtig, in den GCIs keine Konzeptnamen zu verwenden, die in der TBox definiert sind.

Bemerkung: Ob die zyklischen TBoxen oder die primitiven TBoxen jeweils korrekt normalisiert werden, wurde nicht getestet, da dies den externen Programmteilen `gfp.lisp` bzw. `gci.lisp` obliegt.

[2] `gci.lisp` und `gfp.lisp`

4.2 Funktionstests

4.2.1 Test 1

Im ersten Test verwenden wir eine ganz einfache $\mathcal{EL}$-TBox $\mathcal{T}_1$ und eine primitive generelle TBox $\mathcal{V}_1$ mit nur einem primitiven GCI. Die TBox $\mathcal{T}_{1a}$ ist noch nicht nach $\mathcal{V}_1$ vervollständigt; die Transformation sollte zur Definition von AP noch die Existenzrestriktion '$\exists r.AP$' hinzufügen.

Testfall 1 $\mathcal{T}_1$	 $\mathcal{V}_1$
$A_P \equiv P$ $A \equiv \exists r.A$	$P \sqsubseteq \exists r.P$

Tabelle 4.1: Test 1

Testergebnisse für	$\mathcal{T}_1$ und $\mathcal{V}_1$
Erwartetes Ergebnis Funktion: `tbox-completion`	$A_P \equiv P$ $NEW\text{-}DEF_1 \equiv \exists r.A_P$ $A \equiv \exists r.A$ $P \sqsubseteq \exists r.P$
Tatsächliches Ergebnis Funktion: `tbox-completion`	$A_P \equiv P$ $NEW\text{-}DEF_1 \equiv \exists r.A_P$ $A \equiv \exists r.A$ $P \sqsubseteq \exists r.P$
Erwartetes Ergebnis Funktion: `tbox-transformation`	$A_P \equiv P \sqcap \exists r.A_P$ $NEW\text{-}DEF_1 \equiv \exists r.A_P$ $A \equiv \exists r.A$
Tatsächliches Ergebnis Funktion: `tbox-transformation`	$A_P \equiv P \sqcap \exists r.A_P$ $NEW\text{-}DEF_1 \equiv \exists r.A_P$ $A \equiv \exists r.A$

Tabelle 4.2: Testergebnisse 1

Das Programm liefert das erwartete Ergebnis. Durch die Vervollständigung wurde eine neue Konzeptdefinition zu $\mathcal{T}$ hinzugefügt; die Transformation hat die Definition von A_P um die Existenzrestriktion $\exists r.A_P$ erweitert.

4.2.2 Test 2

Die TBox $\mathcal{T}_2$ ist aus [3] (siehe Abschnitt 3 ebd.) übernommen — damit haben wir eine etwas komplexere TBox, die vervollständigt und transformiert werden muss. Die primitive TBox ist von geringer Komplexität, getestet werden soll vor allem, ob die Vervollständigung tatsächlich alle primitiven Konzepte aus $\mathcal{T}_2 \cup \mathcal{V}_2$ erkennt und sofern nötig eine neue Konzeptdefinition dafür erzeugt.

Testfall 2	
$\mathcal{T}_2$	$A_1 \equiv P_1 \sqcap P_2 \sqcap P_3 \sqcap \exists r_1.B_1 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$ $B_1 \equiv \exists r_2.A_2$ $A_2 \equiv P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$ $B_3 \equiv \exists r_1.A_1$ $A_3 \equiv P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$ $B_3 \equiv P_2 \sqcap P3$
$\mathcal{V}_2$	$P_1 \sqcap P_2 \sqsubseteq Q$ $Q \sqsubseteq \exists r1.Q$

Tabelle 4.3: Test 2

Testergebnisse für $\mathcal{T}_2$ und $\mathcal{V}_2$		
		Erwartetes Ergebnis: `tbox-completion`
A_1	$\equiv$	$P_1 \sqcap P_2 \sqcap P_3 \sqcap \exists r_1.B_1 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_1	$\equiv$	$\exists r_2.A_2$
A_2	$\equiv$	$P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_3	$\equiv$	$\exists r_1.A_1$
A_3	$\equiv$	$P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_3	$\equiv$	$P_2 \sqcap P3$
$NEW\text{-}DEF_1$	$\equiv$	Q

Testergebnisse	für	$\mathcal{T}_2$ und $\mathcal{V}_2$
$NEW\text{-}DEF_2$	$\equiv$	P_1
$NEW\text{-}DEF_3$	$\equiv$	P_2
$NEW\text{-}DEF_4$	$\equiv$	P_3
$NEW\text{-}DEF_5$	$\equiv$	$\exists r_1.NEW\text{-}DEF_1$
	—	
$P_1 \sqcap P_2$	$\sqsubseteq$	Q
Q	$\sqsubseteq$	$\exists r1.Q$
		Tatsächliches Ergebnis: `tbox-completion`
A_1	$\equiv$	$P_1 \sqcap P_2 \sqcap P_3 \sqcap \exists r_1.B_1 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_1	$\equiv$	$\exists r_2.A_2$
A_2	$\equiv$	$P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_3	$\equiv$	$\exists r_1.A_1$
A_3	$\equiv$	$P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_3	$\equiv$	$P_2 \sqcap P3$
$NEW\text{-}DEF_1$	$\equiv$	Q
$NEW\text{-}DEF_2$	$\equiv$	P_1
$NEW\text{-}DEF_3$	$\equiv$	P_2
$NEW\text{-}DEF_4$	$\equiv$	P_3
$NEW\text{-}DEF_5$	$\equiv$	$\exists r_1.NEW\text{-}DEF_1$
	—	
$P_1 \sqcap P_2$	$\sqsubseteq$	Q
$P_2 \sqcap P_1$	$\sqsubseteq$	Q
Q	$\sqsubseteq$	$\exists r1.Q$
Bemerkung:		Die Funktion `tbox-completion` erzeugt die GCIs $P_1 \sqcap P_2 \sqsubseteq Q$ sowie $P_2 \sqcap P_1 \sqsubseteq Q$ abweichend von der Erwartung; es ist nicht in der Lage, festzustellen, dass die beiden GCIs redundant sind.
		Erwartetes Ergebnis: `tbox-transformation`
A_1	$\equiv$	$P_1 \sqcap P_2 \sqcap P_3 \sqcap Q \exists r_1.B_1$ $\sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3 \sqcap \exists r_1.NEW\text{-}DEF_1$

Testergebnisse	für	$\mathcal{T}_2$ und $\mathcal{V}_2$
B_1	$\equiv$	$\exists r_2.A_2$
A_2	$\equiv$	$P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_3	$\equiv$	$\exists r_1.A_1$
A_3	$\equiv$	$P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_3	$\equiv$	$P_2 \sqcap P3$
$NEW\text{-}DEF_1$	$\equiv$	Q
$NEW\text{-}DEF_2$	$\equiv$	P_1
$NEW\text{-}DEF_3$	$\equiv$	P_2
$NEW\text{-}DEF_4$	$\equiv$	P_3
$NEW\text{-}DEF_5$	$\equiv$	$\exists r_1.NEW\text{-}DEF_1$
		Tatsächliches Ergebnis:
		`tbox-transformation`
A_1	$\equiv$	$P_1 \sqcap P_2 \sqcap P_3 \sqcap Q \exists r_1.B_1$
		$\sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3 \sqcap \exists r_1.NEW\text{-}DEF_1$
B_1	$\equiv$	$\exists r_2.A_2$
A_2	$\equiv$	$P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_3	$\equiv$	$\exists r_1.A_1$
A_3	$\equiv$	$P_2 \sqcap P_3 \sqcap \exists r_2.B_2 \sqcap \exists r_1.B_3$
B_3	$\equiv$	$P_2 \sqcap P3$
$NEW\text{-}DEF_1$	$\equiv$	Q
$NEW\text{-}DEF_2$	$\equiv$	P_1
$NEW\text{-}DEF_3$	$\equiv$	P_2
$NEW\text{-}DEF_4$	$\equiv$	P_3
$NEW\text{-}DEF_5$	$\equiv$	$\exists r_1.NEW\text{-}DEF_1$

Tabelle 4.4: Testergebnisse 2

Die Vervollständigung ist zwar nicht in der Lage die Redundanz der GCIs $P_2 \sqcap P_3 \sqsubseteq Q_1$ und $P_3 \sqcap P_2 \sqsubseteq Q_1$ zu erkennen. Dadurch wurde jedoch das Ergebnis der `tbox-transformation` nicht beeinträchtigt. Wegen der Kommutativität des ‚⊓'-Operators ist dies auch nicht zu erwarten und es ist anzunehmen, dass dieses Problem eher in den Bereich der Laufzeitoptimierung fällt. Zudem liegt die Ursache dieses Effekts in der Datenstruktur `*gci-hash*`, welche in der externen

Datei `gci.lisp` definiert ist. Das Problem — sofern es als ein solches angesehen wird — sollte eventuell eher dort behandelt werden.

4.2.3 Bewertung der Tests

Abgesehen von der Erzeugung von redundanten GCIs durch die Vervollständigung konnten durch diese beiden Tests keine Fehlfunktion des Programms nachgewiesen werden. Natürlich reichen zwei Tests noch lange nicht aus, um hinreichend sicher zu sein, dass das Programm einigermaßen fehlerfrei ist. Es wurde z.B. nicht getestet, wie sich das Programm bei falsch formatierten Eingaben verhält. Außerdem sollten systematische Tests von anderer Seite her durchgeführt werden, als vom Entwickler selbst.

Kapitel 5

Schluss

Betrachtet man zyklische $\mathcal{EL}$-TBoxen mit *gfp*-Semantik, so ist garantiert, dass das lcs existiert und berechnet werden kann [7]. Für generelle $\mathcal{EL}$-TBoxen ist dies nicht möglich, da GCIs mit *gfp*-Semantik wenig Sinn ergeben. Um trotzdem GCIs benutzen zu können, ohne die *gfp*-Semantik aufzugeben, wurden s.g. hybride $\mathcal{EL}$-TBoxen konzipiert, in der sowohl zyklische Definitionen vorkommen, die mit *gfp*-Semantik betrachtet werden, als auch GCIs, die mit deskriptiver Semantik betrachtet werden. Wir haben gezeigt, dass das Subsumtionsproblem für diese hybriden $\mathcal{EL}$-TBoxen mit *gfp*-Semantik in polynomieller Zeit auf das Subsumtionsproblem für zyklische TBoxen ohne die s.g. primitiven GCIs reduziert werden kann und haben unter Verwendung der Ergebnisse aus [9] und [3] einen Algorithmus dafür angegeben. Dies erlaubt es, die nützlichen Nicht-Standard-Inferenzen auch für Wissensbasen anzubieten, die GCIs verwenden, und wir sind überzeugt, dass sich dafür sinnvolle Anwendungen finden werden.

Außerdem haben wir einen prototypischen *Reasoner* in der Sprache LISP für die Berechnung der *gfp*-Subsumtion für hybride $\mathcal{EL}$-TBoxen entwickelt sowie die Ergebnisse von Funktionstests dokumentiert. Der Hybrid-$\mathcal{EL}$-Reasoner basiert auf den Implementationen für den Subsumtionsalgorithmus für generelle $\mathcal{EL}$-TBoxen und dem Subsumtionsalgorithmus für zyklische $\mathcal{EL}$-TBoxen mit *gfp*-Semantik. Beide Programme wurden im Rahmen einer ‚*Master's Thesis*'an der TU Dresden entwickelt und in [22] vorgestellt. Die Dokumentation der Datenstrukturen und Funktionen des Hybrid-$\mathcal{EL}$-Reasoner sowie der kommentierte Quellcode zum Hybrid-$\mathcal{EL}$-Reasoner befinden sich im Anhang. Es sollte

anderen LISP-Programmieren möglich sein, den Hybrid-$\mathcal{EL}$-Reasoner weiter zu entwickeln, und/oder ihn in andere *Reasoner* zu integrieren, um ihn für reale Anwendungen zum Einsatz zu bringen.

Literaturverzeichnis

[1] F. Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90*, pages 621–626, Boston (USA), 1990.

[2] F. Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):175–219, 1996.

[3] F. Baader. Terminological cycles in a description logic with existential restrictions. LTCS-Report LTCS-02-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2002. See http://lat.inf.tu-dresden.de/research/reports.html.

[4] F. Baader and W. Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

[5] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[6] Franz Baader. Computing the least common subsumer in the description logic $\mathcal{EL}$ w.r.t. terminological cycles with descriptive semantics. In *Proceedings of the 11th International Conference on Conceptual Structures, ICCS 2003*, volume 2746 of *Lecture Notes in Artificial Intelligence*, pages 117–130. Springer-Verlag, 2003.

[7] Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In

Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 319–324. Morgan Kaufmann, 2003.

[8] S. Brandt and A.-Y. Turhan. Using non-standard inferences in description logics — what does it buy me? In *Proceedings of the KI-2001 Workshop on Applications of Description Logics (KIDLWS'01)*, number 44 in CEUR-WS, Vienna, Austria, September 2001. RWTH Aachen. Proceedings online available from http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-44/.

[9] Sebastian Brandt. On subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general tboxes. In *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, CEUR-WS, 2004.

[10] Italo Calvino. *Die unsichtbaren Städte.* Deutscher Taschenbuch Verlag GmbH&Co.KG, München, 1985.

[11] The Gene Ontology^{TM} Consortium. http://www.geneontology.org.

[12] R.A. Cote, D.J. Rothwell, J.L.Palotey, R.S. Beckett, and L. Brochu. The systematized nomenclature of human and veterinary medicine. Technical report, SNOMED International, Northfield, IL: College of American Pathologists, 1993.

[13] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In IEEE, editor, *36th Annual Symposium on Foundations of Computer Science: October 23–25, 1995, Milwaukee, Wisconsin*, pages 453–462, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995. IEEE Computer Society Press.

[14] I. Horrocks, A.L. Rector, and Carole A. Goble. A description logic based schema for the classification of medical data. In Franz Baader, Martin Buchheit, Manfred A. Jeusfeld, and Werner Nutt, editors, *Knowledge Representation meets Databases, Proceedings of the 3rd Workshop KRDB'96, Budapest, Hungary.* Technical University of Aachen (RWTH), 1996.

[15] I. Horrocks and U. Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th European Conference on Artificial Intelligence*, 2002.

[16] Ian R. Horrocks. Using an expressive description logic: FaCT or fiction? In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 636–645. Morgan Kaufmann, San Francisco, California, 1998.

[17] B. Nebel. Terminological cycles: Semantics and computational properties. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991.

[18] CMU Common Lisp Project. CMUCL. http://www.cons.org/cmucl/.

[19] A. Rector, W. Nolan, and A. Glowinski. Goals for concept representation in the GALEN project. In *Proceedings of the 17th annual Symposium on Computer Applictions in Medical Care, Washington, USA, SCAMC*, pages 414–418, 1993.

[20] Ulrike Sattler. *Terminological knowledge representation systems in a process engineering application.* PhD thesis, LuFG Theoretical Computer Science, RWTH Aachen, 1998.

[21] J.F. Sowa. Semantic networks. In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligenz 2.* John Wiley & Sons, New York, 1987.

[22] B. Suntisrivaraporn. Implementation and Optimization of Subsumption Algorithms in the DL $\mathcal{EL}$ with Cyclic TBoxes and General Concept Inclusion Axioms. Master's thesis, Dresden University of Technology, Germany, 12 2004. See http://lat.inf.tu-dresden.de/research/reports.html.

[23] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

Anhang A

Dokumentation des Hybrid-$\mathcal{EL}$-Reasoner

Die Datei `hybrid-reasoner.lisp` enthält

Programmzeilen	248
Kommentarzeilen	144
Leerzeilen	36
Gesamt	426

A.1 Datenstrukturen

Wir geben hier eine Dokumentation aller global sichtbaren Datenstrukturen - Konstanten und Variablen - die in `hybrid-reasoner.lisp` definiert werden. Außerdem werden noch einige wichtige Datenstrukturen aus den Implementierungen des Subsumtionsalgorithmus für generelle $\mathcal{EL}$-TBoxen— siehe Datei `gci.lisp` — sowie der Implementierungen des Subsumtionsalgorithmus für zyklische $\mathcal{EL}$-TBoxen mit *gfp*-Semantik — siehe Datei `gfp.lisp` — dokumentiert, sofern sie im Hybrid-$\mathcal{EL}$-Reasoner benutzt werden.

- `*def-keyword*` KONSTANTE

  Definiert in: `gci.lisp` und `gfp.lisp`

  Typ: Symbol

Wert: `'DEFCONCEPT` — Schlüsselwort für Konzeptdefinitionen

- `*gci-keyword*` KONSTANTE

  Definiert in: `gci.lisp`

  Typ: Symbol

  Wert: `'IMPLIES` — Schlüsselwort für GCIs.

- `*and-keyword*` KONSTANTE

  Definiert in: `gci.lisp` und `gfp.lisp`

  Typ: Symbol

  Wert: `'AND` — Schlüsselwort für eine Konjunktion.

- `*some-keyword*` KONSTANTE

  Definiert in: `gci.lisp` und `gfp.lisp`

  Typ: Symbol

  Wert: `'SOME` — Schlüsselwort für eine Existenzrestriktion.

- `*concept-names*` VARIABLE

  Definiert in: `gci.lisp`

  Typ: Liste

  Beschreibung: Liste aller Konzeptnamen, die in einem der GCIs auftauchen.

- `*gci-tbox*` VARIABLE

  Definiert in: `gci.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der normalisierten GCIs.

  Schlüssel: Konzept `P` oder Existenzrestriktion `(SOME R P)`

  Wert: Wenn der Schlüssel eine Existenzrestriktion `(SOME R P)` ist, dann kann der Wert nur ein Konzeptname `Q` sein. Wenn der Schlüssel ein Konzeptname `P` ist, kann der Wert entweder eine Existenzrestriktion `(SOME R Q)` sein oder ein *dotted pair* `(Q1.Q2)` sein — dies ist die Kodierung für ein GCI der Form $(P \sqcap Q_1) \sqsubseteq Q_2$.

- `*s*` VARIABLE

  Definiert in: `gci.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der Subsumtionsmengen.

  Schlüssel: Konzeptname `A`

  Wert: Liste von Konzeptnamen, die `A` subsumieren.

- `*graph-label-hash*` VARIABLE

  Definiert in: `gfp.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der Labelmengen der Knoten im $\mathcal{EL}$-Beschreibungsgraphen — entspricht den primitiven Konzeptnamen auf der rechten Seite einer Konzeptdefinition.

  Schlüssel: Konzeptname `A`

  Wert: Liste von primitiven Konzeptnamen.

- `*graph-out-edge-hash*` VARIABLE

  Definiert in: `gfp.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der ausgehenden Kanten im $\mathcal{EL}$-Beschreibungsgraphen — entspricht den Existenzrestriktionen auf der rechten Seite einer Konzeptdefinition.

  Schlüssel: Ein *dotted pair* `(A.R)` — die Kombination aus Knotenname (bzw.definiertes Konzept) und Kantenlabel (bzw. Rollenname)

  Wert: Liste von Knotennamen, in denen die mit `R` gelabelten Kanten enden, die in von `A` ausgehen.

- `*graph-in-edge-hash*` VARIABLE

  Definiert in: `gfp.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der eingehenden Kanten im $\mathcal{EL}$-Beschreibungsgraphen.

  Schlüssel: Ein *dotted pair* `(A.R)` — die Kombination aus Knotenname

(bzw.definiertes Konzept) und Kantenlabel (bzw. Rollenname)

Wert: Liste von Knotennamen, von denen die mit `R` gelabelten Kanten ausgehen, die in `A` enden.

- `*tbox-hash*` VARIABLE

  Definiert in: `gfp.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der TBox.

  Schlüssel: Ein Konzeptname `A`.

  Wert: Konzeptbeschreibung der rechten Seite der Definition von `A`.

- `*defined-concepts*` VARIABLE

  Definiert in: `gfp.lisp`

  Typ: Liste

  Beschreibung: Datenstruktur zur Repräsentation der definierten Konzepten der TBox.

- `*new-concept-name*` KONSTANTE

  Definiert in: `hybrid-reasoner.lisp`

  Wert : `"NEW-DEF-"`

  Typ: String

  Beschreibung: Ein Präfix für die Generierung neuer Symbole für definierte Konzepte.

- `*map-prim-to-def-hash*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: Hashtabelle

  Beschreibung: Ordnet den primitiven Konzepten aus der TBox und der primitiven TBox einen definierten Konzeptnamen zu.

  Schlüssel: Ein primitiver Konzeptname `P`.

  Wert: Definiertes Konzept `AP`, für das gilt `(DEFCONCEPT AP P)`.

- `*primitive-concepts*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

Typ: Liste

Beschreibung: Liste der primitiven Konzepte aus der TBox und der primitiven TBox.

- `*gfp-subsumption-set-hash*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der Subsumtionsbeziehungen in der transformierten TBox. Diese Datenstruktur ist möglicherweise redundant und wird auch nicht in den Kernfunktionen des Hybrid-$\mathcal{EL}$-Reasoners verwendet.

  Schlüssel: Ein Konzeptname `A`.

  Wert: Liste von Konzepten, die `A` subsumieren.

- `*completed-tbox-hash*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der vervollständigten Tbox.

  Schlüssel: Ein Konzeptname `A`.

  Wert: Liste von Konzepten, die in der (normalisierten) Definition von `A` vorkommen. Also entweder primitive Konzepte oder Existenzrestriktionen mit definierten Konzepten.

- `*normalized-gcis*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: Liste

  Beschreibung: Liste von normalisierten GCIs, wie sie an die Datei der vervollständigten TBox angehängt werden.

- `*tbox-representation-hash*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: Hashtabelle

  Beschreibung: Datenstruktur zur Repräsentation der transformierten

Tbox.

Schlüssel: Ein Konzeptname `A`.

Wert: Liste von Konzepten, die in der (normalisierten) Definition von `A` vorkommen. Also entweder primitive Konzepte oder Existenzrestriktionen mit definierten Konzepten.

- `*transformed-tbox-file*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: String

  Beschreibung: Name der Datei, in die transformierte TBox geschrieben wird.

- `*completed-tbox-file*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: String

  Beschreibung: Name der Datei, in die vervollständigte TBox geschrieben wird.

- `*tbox*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: String

  Beschreibung: Name der Datei in der die TBox gespeichert ist.

- `*vbox*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: String

  Beschreibung: Name der Datei in der die primitive TBox gespeichert ist.

- `*debug*` VARIABLE

  Definiert in: `hybrid-reasoner.lisp`

  Typ: Symbol

  Beschreibung: Globales *Debug-Flag.*

A.2 Funktionen

Bei der Dokumentation Funktionen beschränken wir uns auf die in der Datei `hybrid-reasoner.lisp` definierten Funktionen, bis auf drei für uns wichtige Ausnahmen.

- `d-start` FUNKTION

  Definiert in: `gci.lisp`

  Parameter: `file-name` — ein String mit dem Namen der Datei, in der die generelle TBox gespeichert ist.

  Rückgabewert: Bei Erfolg: `SUCCESS` andernfalls `FAILED`.

  Bemerkung: Name wurde geändert um Nameskollisionen zu vermeiden

- `start` FUNKTION

  Definiert in: `gfp.lisp`

  Parameter: `file-name` — ein String mit dem Namen der Datei, in der die zyklische TBox gespeichert ist.

  Rückgabewert: Bei Erfolg: `SUCCESS` andernfalls `FAILED`.

- `gfp-subsumes` FUNKTION

  Definiert in: `gfp.lisp`

  Parameter: `'A 'B` — zwei Symbole `'A` und `'B` aus der Liste der `*defined-concepts*`

  Rückgabewert: Wenn `'A` von `'B` subsumiert wird gibt die Funktion `'YES` zurück, wenn `'A` nicht von `'B` subsumiert wird, `'NO`. Falls eines der Symbole nicht in `*defined-concepts*` enthalten ist: `'UNKNOWN`

  Bemerkung: Name wurde geändert um Nameskollisionen zu vermeiden

Alle anderen Funktionen, die wir noch dokumentieren, stammen aus der Datei `hybrid-reasoner.lisp`.

- `init-hybrid-reasoner` FUNKTION

  Beschreibung: Selbsterklärende Bezeichnung.

  Parameter:

  Rückgabewert: `T`

- `generate-new-concept-name` FUNKTION

  Beschreibung: Erzeugt ein neues Symbol.

  Parameter:

  Rückgabewert: Ein neues Symbol `NEW-DEF-X`, wobei 'X' die fortlaufende Nummer neu generierten Symbols ist.

- `get-values` FUNKTION

  Beschreibung: Liefert die Werte einer Hastabelle als Liste — zur Zeit noch nicht benutzt.

  Parameter: `HASH-TABLE` — eine Hashtabelle

  Rückgabewert: Liste der in der Hashtabelle gespeicherten Werte.

- `gfp-subsumption-tbox-init` FUNKTION

  Beschreibung: Intern benutzt, um die Hashtabelle `*gfp-subsumption -set-hash*` initial mit Schlüssel-Wert Paaren zu füllen.

  Parameter:

  Rückgabewert: `NIL`

- `primitive?` FUNKTION

  Beschreibung: Überprüft, ob ein Konzept ein primitives Konzept ist.

  Parameter: `concept` — ein Konzeptname.

  Rückgabewert: `T`, wenn `concept` ein primitives Konzept ist, andernfalls `NIL`.

- `defined?` FUNKTION

  Beschreibung: Überprüft, ob ein Konzept ein definiertes Konzept ist.

  Parameter: `concept` — ein Konzeptname.

  Rückgabewert: `T`, wenn `concept` ein definiertes Konzept ist, andernfalls `NIL`.

- `map-prim-to-def` FUNKTION

  Beschreibung: Liefert zu einem primitiven Konzeptnamen den korrespondierenden definierten Namen.

  Parameter: `P` — ein primitiver Konzeptname

Rückgabewert: Der definierte Konzeptname `AP` aus der Hashtabelle `*map-prim-to-def-hash*`. Wenn `P` noch keinen korrespondierenden definierten Namen hat `Nil`.

- `get-primitives-from-tbox` FUNKTION

  Beschreibung: s.U.

  Parameter:

  Rückgabewert: Liste der primitiven Konzepte, die in der TBox (nicht in der primitiven TBox) vorkommen.

- `tbox-completion` FUNKTION

  Beschreibung: Wird in der Funktion `run` aufgerufen — selbsterklärender Name.

  Parameter: `tbox vbox` - eine TBox `tbox` und eine primitive TBox `vbox`

  Rückgabewert: `T`

- `completion-kernel` FUNKTION

  Beschreibung: Zentrale Funktion zur Berechnung der vervollständigten TBox— wird intern von `tbox-completion` aufgerufen.

  Parameter:

  Rückgabewert: `T`

- `check-some-clause` FUNKTION

  Beschreibung: Überprüft, ob für die Existenzrestriktion `(SOME role-name defined-concept)` eine neue Konzeptdefinition gebraucht wird — wird in der Function `tbox-completion` gebraucht.

  Parameter: `role-name defined-concept` — Ein Rollenname und ein Konzeptname

  Rückgabewert: Falls ja, gibt die Funktion den neuen Konzeptnamen zurück, nachdem alle wichtigen Datenstrukturen entsprechend aktualisiert wurden. Falls nein, gibt die Funktion `NIL` zurück.

- `write-out-completed-tbox` FUNKTION

Beschreibung: Schreibt die vervollständigte TBox in eine Datei.

Parameter: `c-tbox-file` - Name der Datei, in die vervollständigte TBox geschrieben wird.

Rückgabewert: `NIL`

- `norm-gcis` FUNKTION

  Beschreibung: Initialisiert eine Liste mit den normalisierten GCIs aus der primitiven TBox, die an die vervollständigte TBox angehängt wird.

  Parameter:

  Rückgabewert: `NIL`

- `tbox-transformation` FUNKTION

  Beschreibung: Für jedes Konzept `next` aus `*defined-concepts*` und jedes Konzept `concept` aus der Subsumtionsmenge `*s*` von `next` wird die Funktion `tbox-transformation-kernel` aufgerufen. Abschließend wird die Funktion `write-out-transformed-tbox` aufgerufen.

  Parameter:

  Rückgabewert: `NIL`

- `tbox-transformation-kernel` FUNKTION

  Beschreibung: Wichtigste Funktion zur Transformation der TBox. Aktualisiert die Hashtabelle `*tbox-representation-hash*`.

  Parameter: `sub-concept super-concept` — Zwei Konzeptnamen.

  Rückgabewert: `T`

- `write-out-transformed-tbox` FUNKTION

  Beschreibung: Schreibt die transformierte TBox in eine Datei mit dem Namen `"transformed-tbox.lisp"` im gleichen Verzeichnis, in dem der Hybrid-$\mathcal{EL}$-Reasoner gestartet wurde.

  Parameter:

  Rückgabewert: `T`

- `run` FUNKTION

  Beschreibung: Einzige wirklich als *public* konzipierte Funktion — ruft

zunächst die Funktion `tbox-completion` auf. Danach wird die Funktion `d-start` aus `gci.lisp` mit vervollständigte TBox aufgerufen. Nach der Berechnung der Subsumtionsmengen wird die transformierte TBox erzeugt und in der Datei `transformed-tbox.lisp` gespeichert. Mit dem Namen dieser Datei schließlich wird die Funktion `gfp-start` aus `gfp.lisp` aufgerufen und die *gfp*-Subsumtion für die transformierte TBox berechnet.

Parameter: `tbox vbox` &key `:debug` - Strings mit den Namen der TBox und der primitiven TBox und optional, wenn mehr Meldungen auf der Konsole gewünscht sind `:debug t`.

Rückgabewert: `NIL`

Anhang B

Quellcode

```
;;;; ============================================================================
;;;;
;;;; Diese Datei ist Teil der Diplomarbeit von  Jörg Model , MtrNr: 2634548
;;;; und ist dort im 'verbatim' im Appendix zu finden.
;;;;
;;;;
;;;;
;;;; Titel der Diplomarbeit ::  Subsumtion in EL bezüglich hybrider TBoxen
;;;; Author                 ::  Jörg Model
;;;; Matrikelnummer         ::  2634548
;;;; Jahr                   ::  2005
;;;; Universität            ::  TU Dresden
;;;; Institut               ::  Lehrstuhl für Automatentheorie, Fakultät Informatik
;;;; Hochschulprofessor     ::  Prof. Dr. Ing Franz Baader
;;;; Betreuer               ::  Dipl. Inf. Sebastian Brandt
;;;;
;;;; Kommentar              ::  All comments are in English to facilitate the
;;;;                            reusability for non german developers. Moreover I
;;;;                            tried to use names which are self-explaining. The
;;;;                            policy for naming is: '*variable*' indicates, that
;;;;                            this is global varibale while 'variable' is used in
;;;;                            a local context.
;;;; ============================================================================
;;;; This is an implementation of the gfp-subsumtion algorithm for EL-tboxes with
;;;; aditional primitive GCIs, so called hybrid EL-TBOXES.
;;;;
;;;;
;;;; Created: 22.11.2004
;;;; Last Modified:13.02.2005
;;;; IDE: CMU-Lisp and Emacs on a Linux machine
;;;; Tested runtime system: CMU-LISP
;;;;============================================================================

;;; *NEW-CONCEPT-NAME*  a prefix needed as for the function 'gentemp' -
;;;  for description type:   (describe 'gentemp)   in a running lisp interpreter
(defconstant *new-concept-name* "NEW-DEF-")

;;; *PRIMITIVE-CONCEPTS*  stores primitive concepts.
(defvar *primitive-concepts* nil)
(defvar *map-prim-to-def-hash* nil)

;;; *GFP-SUBSUMPTION-SET-HASH*:
;;; Each defined concept name 'A' is associated to a set of
;;; concept names (primitives and defined), which do subsume A
;;; w.r.t. a given hybrid tbox with \gfp-semantic.
;;; key::  concept name - defined
;;; value: a list concept names - defined or primitive
(defvar *gfp-subsumption-set-hash* nil)

;;; *COMPLETED-TBOX-HASH*: representation of the completed TBox
;;;  - mainly used for writing out the completed tbox
```

```
;;; key::  concept name - defined
;;; value: a list concept names - defined or primitive
(defvar *completed-tbox-hash* nil)

;;; *NORMALIZED-GCIS*: representation of the normalized
;;; primitive general TBox. Used for Writing the completed tbox
;;; DATA:: list of gcis
(defvar *normalized-gcis* nil)

;;; *TBOX-REPRESENTATION-HASH*: representation of the TBox
;;; mainly used for the transformation and writing the *transformed-tbox-file*
(defvar *tbox-representation-hash* nil)

;;; File name  storing the transformed TBox
(defvar *transformed-tbox-file* nil)

;;; File name  storing the completed TBox
(defvar *completed-tbox-file*  nil)

;;;; Global Debug-Flag
(defvar *debug* nil)

;;;; self explaining names
(defvar *tbox* nil)
(defvar *vbox* nil)

;;;; IMPORTANT :: We make use of extern datastructures defined in 'gci.lisp' and
;;;; 'gfp.lisp'. These two files implement the subsumption for general EL TBoxes
;;;; with descriptive semantics and gfp-subsumption for cyclic EL TBoxes.
;;;; Unfortunatly we could not just take  these files without modifing them slightly
;;;; because of namesspace collisions (LISP does not provide satisfiing features
;;;; with respect to that). This means, one cannot just take a new version of
;;;; that files together with the hybrid EL reasoner!!

;; ============================================================================== ;;
;;
;;      SECTION:: MISC FUNCTIONS
;;
;; ============================================================================== ;;
(defun init-hybrid-reasoner ()
  (setf *normalized-gcis* nil)
  (setf *primitive-concepts* nil)
  (setf  *transformed-tbox-file*  "transformed-tbox.lisp")
  (setf  *completed-tbox-file*   "completed-tbox.lisp")
  (setf *map-prim-to-def-hash* (make-hash-table :test 'equal))
  (setf *completed-tbox-hash* (make-hash-table :test 'equal))
  (setf *tbox-representation-hash* (make-hash-table :test 'equal))
  (setf  *gfp-subsumption-set-hash* (make-hash-table :test 'equal))
  T
  )


;; ------------------------------------------------------------------------ ;;
(defun generate-new-concept-name ()
   (gentemp *new-concept-name*))

;; ------------------------------------------------------------------------ ;;
;; recently not used.
(defun get-values (hash-table)
  (let ((values nil))
    (maphash #'(lambda (key value) (push value values)) hash-table)
    values))


;; ------------------------------------------------------------------------ ;;
(defun gfp-subsumption-tbox-init ()
  (dolist (def-name1 *defined-concepts* nil)
```

```
    (let ((subs-list (gethash def-name1 *s*)))
    (dolist (def-name2 *defined-concepts* nil)
      (when (eql 'YES (progn (gfp-subsumes? def-name2 def-name1)))
        (unless (member def-name2 subs-list :test 'equal)
          (setf  subs-list (cons def-name2 subs-list ))
        )))
    (setf (gethash def-name1 *gfp-subsumption-set-hash*) subs-list)))
  )

;; ---------------------------------------------------------------------- ;;
(defun primitive? (concept) (member concept *primitive-concepts*))

;; ---------------------------------------------------------------------- ;;
(defun defined? (concept) (member concept  *defined-concepts* ))

;; ---------------------------------------------------------------------- ;;
(defun map-prim-to-def (p) (gethash p *map-prim-to-def-hash* ))


;; ---------------------------------------------------------------------- ;;
(defun get-primitives-from-tbox()
  (let ((primitives-list nil))
    ;; *defined-concepts* is a list defined defined in 'gfp.lisp'
    (dolist (defined-concept  *defined-concepts* primitives-list)
      (let ((prim-list  (gethash defined-concept *graph-label-hash*) ))
        ;; if possible, add a piece of information to the hash table, which
        ;; maps primitive concepts to their defined equivalents
        (if (eql (length prim-list) 1)
            (setf (gethash (car prim-list)
                           *map-prim-to-def-hash*) defined-concept))
        (dolist (primitive-concept  prim-list nil)
          (unless (member  primitive-concept  primitives-list :test 'equal)
            (setf primitives-list (cons primitive-concept primitives-list)))
          )))))

;; ================================================================  ;;
;;
;;          SECTION :: TBOX-COMPLETION
;; ================================================================  ;;
;;
;; TBOX-COMPLETION: reading a  tbox and a primitive tbox from files,
;; completing the  tbox with defined names for all  primitive concepts
;; names occurring in the  primitive Tbox and for all existencial
;; restrictions. And writes them out to a file in normailized form.
;; ----------------------------------------------------------------  ;;
(defun tbox-completion (tbox vbox )
  (init-hybrid-reasoner)
  ;; call 'start' from 'gfp.lisp' to init the neccesary
  ;; data-structures in 'gfp.x86f' - especially the hash-tables
  ;; '*graph-edge-in-hash*' ,  '*graph-edge-out-hash*' and
  ;; '*graph-label-hash*' (completion) and '*tbox-hash*' (transformation)
  (start tbox)
  (setf *primitive-concepts* (get-primitives-from-tbox))
  ;; initialize the data-structures in gci.x86f with the vbox - especially
  ;; we need '*concept-names*' and  '*gci-tbox*'  (completion)
  (d-start vbox)
  ;; all primitive concepts should have a defined equivalent!
  (dolist (p-concept  *concept-names* *primitive-concepts*)
    (if (not (member p-concept *primitive-concepts* :test 'equal))
        (setf *primitive-concepts*
              (cons p-concept *primitive-concepts* ))))
  (norm-gcis)
  (completion-kernel)
  (write-out-completed-tbox *completed-tbox-file*)
  T
  )
```

```
;; ---------------------------------------------------------------  ;;
(defun completion-kernel()
  ;; if there is still  a primitive concept, which don't have a defined
  ;; equivalent,  add a new defined concept for it.
  (dolist (pc *primitive-concepts* t)
    (when *debug* (format t "~%<HYBRID-REASONER> completion-kernel: ~S~%"
                          pc ))
    (if  (eql nil (map-prim-to-def pc))
        (let ((new-concept (generate-new-concept-name)))
          (setf (gethash pc *map-prim-to-def-hash*) new-concept)
          (setf (gethash new-concept *graph-label-hash*) (cons pc nil))
          (setf *defined-concepts* (cons new-concept *defined-concepts*))
          (setf (gethash new-concept *tbox-hash*) pc)
          )))
  ;; Check now all keys from *gci-tbox* (defined in gci.lisp) ;
  ;; 'get-keys' is also defined in 'gci.lisp'.
  (dolist (primitive-key (get-keys *gci-tbox*) nil)
     ;;; ... if it is a SOME-CLAUSE ..
    (when (listp primitive-key)
      (if *debug* (format t "~%<HYBRID-REASONER> completion-kernel: ~S~%"
                          primitive-key))
      (let ((role-name (cadr primitive-key))
            (defined-concept (map-prim-to-def (caddr  primitive-key))))
        ;; ... check, if we need a new defined concept for the SOME-CLAUSE
        ;; with equivalent defined concept.
            (check-some-clause role-name defined-concept)));; 'when' ends here
     ;; Do the analog for all values  for the primtive-keys from *gci-tbox*
     (dolist (primitive-value (gethash  primitive-key  *gci-tbox*) nil)
       (when (listp  primitive-value)
         ;; if it is a dotted pair
         (if (not (atom (cdr primitive-value)))
           (let ((role-name (cadr primitive-value))
                 (defined-concept (gethash (caddr  primitive-value)
                                           *map-prim-to-def-hash*)))
             (check-some-clause role-name defined-concept))
           ));; when ends here
       ))
   ;; now construct the *completed-tbox-hash*
   (dolist (defined-concept *defined-concepts* nil)
     (let ((right-hand-side (gethash defined-concept *graph-label-hash*)))
       (maphash #'(lambda (key value)
                    (if (member defined-concept value :test 'equal)
                        (setf  right-hand-side
                               (cons (list *some-keyword* (cdr key) (car key))
                                     right-hand-side))
                      )) *graph-in-edge-hash*)
       (cond ((< 1 (length right-hand-side))
              (setf right-hand-side (cons *and-keyword* right-hand-side)))
             ((eql 1 (length right-hand-side))
              (setf right-hand-side (car right-hand-side))))
       (setf (gethash defined-concept *completed-tbox-hash*) right-hand-side)
       ));
   (when *debug*
     (format t "~%<HYBRID-REASONER> Completed tbox has following structure:")
     (print-hash *completed-tbox-hash*))
   T
   )

;; ---------------------------------------------------------------%
(defun check-some-clause (role-name defined-concept)
  (if (eql nil (gethash (cons defined-concept role-name) *graph-in-edge-hash*))
      (let ((new-concept (generate-new-concept-name)))
        (setf *defined-concepts* (cons new-concept *defined-concepts*))
        (setf (gethash (cons new-concept role-name ) *graph-out-edge-hash*)
              (cons defined-concept nil ))
        (setf (gethash (cons defined-concept role-name) *graph-in-edge-hash*)
              (cons new-concept nil))
```

```
        (setf (gethash new-concept *tbox-hash*)
              (list *some-keyword* role-name  defined-concept))
        ));; if-then-else ends here
  )

;; ---------------------------------------------------------------%
(defun write-out-completed-tbox  (c-tbox-file)
  "Writes the completed tbox to a file named   c-tbox-file in normalised
form and appends the GCIs from the primitive tbox."

  (with-open-file(out-stream  c-tbox-file  :direction :output )
                 (dolist  (next (get-keys  *completed-tbox-hash*) t)
                   (let ((right-side (gethash next *completed-tbox-hash*))
                         (to-file nil))
                     (setf to-file  (cons *def-keyword*
                                          (cons next (cons right-side nil))))
                     (print to-file out-stream )
                     ))
                  (dolist (line *normalized-gcis* t)
                    (print line out-stream)
                    ))
  (format t "~%Wrote the completed tbox to ~S in directory ~S~%"
          c-tbox-file (default-directory))
  (setf *normalized-gcis*      nil)
  (setf  *completed-tbox-hash* nil)
  )

;; ---------------------------------------------------------------%
(defun norm-gcis ()
  (maphash
   #'(lambda (key value-list)
       (cond ((listp key) ;; A SOME-CLAUSE
              (dolist (value value-list t)
                (setf *normalized-gcis*
                      (cons (list *gci-keyword* key value)
                            *normalized-gcis*))))
             ((atom key)
              (dolist (value value-list t)
                (cond ((atom value)
                       (setf *normalized-gcis*
                             (cons (list *gci-keyword* key value)
                                   *normalized-gcis*)))
                      ((listp value)
                       (if(atom (cdr value))
                           (setf *normalized-gcis*
                                 (cons
                                  (list *gci-keyword*
                                        (list *and-keyword* key
                                              (car value)) (cdr value))
                                  *normalized-gcis*))
                         (setf *normalized-gcis*
                               (cons (list *gci-keyword*
                                           key value)
                                     *normalized-gcis*))))
                      ))))) *gci-tbox*)
  )

;; ============================================================  ;;
;;
;;          SECTION :: TBOX-TRANSFORMATION
;; ============================================================  ;;
;;
;; ------------------------------------------------------------  ;;
;; tbox-transformation :: checks for a defined concept 'next' in
;; *defined-concepts* (defined in 'gfp.lisp') if all conceptnames in
;; the subsumtption set of 'next' ( actually the value connected to
;; 'next' in the hash-table  *s* (defined in gci.lisp))  occur in the
```

```
;; list which is associated with 'next' via
;; '(gethash next *tbox-representation-hash*)' and add the concept if not.
;; Finally the data-structure *tbox-representation-hash* is used to
;; write out the transformed tbox to a file.
;; ------------------------------------------------------------  ;;;
(defun tbox-transformation ()
  (dolist (next *defined-concepts* nil)
    (format  *debug*  "<HYBRID-REASONER> Checking defined concept ~S: ~%"
             next)
    (let ((subsumtion-set (gethash next *s*)))
      (dolist (concept subsumtion-set t)
        (tbox-transformation-kernel next concept))))
  (write-out-transformed-tbox )
  )

;; ------------------------------------------------------------  ;;;
(defun tbox-transformation-kernel (sub-concept super-concept)
  (let ((def-of-super nil)
        (def-of-sub  (gethash sub-concept *tbox-representation-hash*)))
    ;;; If it's a primitive concept
    (if (primitive? super-concept)
        (unless (member super-concept def-of-sub :test 'eql)
          (setf   def-of-sub (cons super-concept def-of-sub)))
      ;; else it's a defined concept
      (do()(t(setf def-of-super
                   (gethash super-concept *tbox-hash*))
             (cond
              ((primitive? def-of-super)
               (unless (member def-of-super def-of-sub :test 'eql)
                 (setf def-of-sub (cons def-of-super def-of-sub))))
              ((and (not (eql nil def-of-super)) (listp def-of-super))
               ;; if it is a list witch begins with 'AND ...
               (if (eql *and-keyword* (car def-of-super ))
                   (dolist (con (cdr def-of-super) t)
                     (if (not(member con def-of-sub :test 'equal))
                         (setf def-of-sub (cons con def-of-sub))))
                 ;; ... else it is a single SOME-CLAUSE (should be)
                 (if (not(member def-of-super def-of-sub :test 'equal))
                     (setf def-of-sub (cons def-of-super def-of-sub))))
               )))))
    (setf (gethash sub-concept *tbox-representation-hash*) def-of-sub)
    );let
  T
  );;

;; ------------------------------------------------------------  ;;;
(defun write-out-transformed-tbox ()
  "Writes the transformed tbox to a file named  *transformed-tbox-file*
in normalised form and formated for the use in the the the lisp programm
'gfp.lisp' to compute gfp-subsumtion."

(with-open-file
 (out-stream *transformed-tbox-file*  :direction :output )
   (dolist  (next *defined-concepts* nil)
     (let ((to-file (gethash next *tbox-representation-hash*)))
     (cond ((> (length to-file) 1 )
            (setf to-file (cons  *and-keyword*  to-file))
            (setf to-file (list to-file))))
     (setf to-file (cons next to-file))
     (setf to-file (cons *def-keyword* to-file))
     (print to-file out-stream )
     )))
);;


;; ================================================================ ;;
;;
```

```
;;     SECTION:: PUBLIC FUNCTION
;;
;; ========================================================================= ;;
(defun run ( tbox vbox &key (debug nil))
  "Main function  with  parameter  'tbox' and 'vbox' and an optional parameter
:debug if You like to see many messages."
  (setf *debug* debug)
  (setf *tbox* tbox)
  (setf *vbox* vbox)
  (format  *debug* "<HYBRID-REASONER> Starting tbox-completion~%")
  (tbox-completion *tbox* *vbox*)
  (d-start *completed-tbox-file*)
  (setf *debug* debug)
  (format *debug* "~%<HYBRID-REASONER> Starting transformation~%")
  (tbox-transformation )
  (start *transformed-tbox-file*)
  (gfp-subsumption-tbox-init)
  (format t "~%To perform a subsumption querry, (gfp-subsumes? 'X 'Y)~%")
  (format t "To perform an equivalent querry, (gfp-equivalent? 'X 'Y)~%")
  (format t "To get everthing in one list, (print-hash  *gfp-subsumption-set-hash*)~%")
  T
  )

;;; ============================================================================  ;;;
;;;
;;;  SECTION:: MAIN
;;;
;;; ============================================================================  ;;;
;;; It's better to use precompiled LISP-code, because it's significant faster, but
;;; there may be good reasons to use the *.lisp files instead. For example while
;;; integrating new versions of the gfp-reasoner or the gci-reasoner.
(load "gci.x86f") ;;(load "gci.lisp")
(load "gfp.x86f") ;;(load "gfp.lisp")

(format t "~%______Subsumption Reasoner in EL for hybrid tboxes__________~%")
(format t "~%To perform subsumption reasoning on a EL-Tbox wíth primitive
             EL-Tbox, (run tbox vbox).")
(format t "~%_____________________________________________________________~%")
;
```

Anhang C

Inhalt der CD

```
./documentation/thesis.ps
./documentation/documentation.ps
./lisp/hybrid-resoner.lisp
./lisp/gfp.lisp
./lisp/gfp.x86f
./lisp/gfp.lisp
./lisp/gfp.x86f
./lisp/tbox1.lisp
./lisp/vbox1.lisp
./lisp/tbox2.lisp
./lisp/vbox2.lisp
```