Diplomarbeit

Entscheidbarkeitsprobleme für Uniqueness Constraints

Christian Düntgen[1]
Feldherrnstraße 41
44147 Dortmund

betreut von

Prof. Dr. Joachim Biskup[2]
Lehrstuhl Informatik VI
Universität Dortmund

und

Dr. Hubert Wagner[3]
Lehrstuhl Informatik V
Universität Dortmund

19. Januar 2005

[1]E-Mail: christian.duentgen@uni-dortmund.de
[2]E-Mail: biskup@ls6.informatik.uni-dortmund.de
[3]E-Mail: wagner@ls5.cs.uni-dortmund.de

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Dortmund, den 19. Januar 2005

_____

(Christian Düntgen)

Zusammenfassung

Objektorientierte Datenmodelle können mittels verschiedener formaler Methoden syntaktisch und semantisch beschrieben werden. Zu diesen Formalismen zählen die *Frame-Logic* und *Description Logics*. Zu den wichtigen Klassen von semantischen Bedingungen für Datenbankschemen zählen *Uniqueness Constraints*. Ein *Uniqueness Constraint* besagt, dass bestimmte Werte oder Objekte gleich sind. Zu dieser Klasse von Abhängigkeiten gehören Pfadabhängigkeiten (PFDs), funktionale Abhängigkeiten (FDs) und Schlüssel (*keys*). Die vorliegende Arbeit beschäftigt sich mit der Fragestellung, ob die logische Implikation von semantischen Bedingungen, die auch *Uniqueness Constraints* umfassen, im Formalismus der *Frame-Logic* entscheidbar sind. Es wird versucht zu klären, ob die logische Implikation in der *Frame-Logic* für Pfadabhängigkeiten, Klasseninklusionsabhängigkeiten und Onto-Abhängigkeiten (das sind Erreichbarkeitszusicherungen) entscheidbar ist. Von diesen Abhängigkeiten stehen die Pfadabhängigkeiten im Mittelpunkt der Untersuchung.

Für viele *Description Logics* ist das Implikationsproblem als entscheidbar bekannt. Daher soll das Problem der logischen Implikation semantischer Bedingungen in *Frame-Logic* in dieser anderen Klasse von formalen Beschreibungsmethoden reformuliert und eine Reduktion der Implikation in der *Frame-Logic* auf die Implikation in der *Description-Logic* angegeben werden.

Die Untersuchung ergibt, dass das Problem auf keine der untersuchten *Description Logics* ($\mathcal{DLR}_{reg,ifd}$, `DLClass`, $\mathcal{DLFRD}$ und $\mathcal{ALCQI}_{reg}$) reduziert werden kann, da die Menge der jeweils verfügbaren Konstrukte unzureichend ist. Verfahren wie die Vereinigung oder Fusion der betrachteten *Description Logics* verlaufen ebenfalls erfolglos, da die Klassen der zu modellierenden semantischen Bedingungen die Kombination von miteinander unverträglichen Konstrukten erfordern, was zur Unentscheidbarkeit der Implikation führt.

Abschließend werden Überlegungen dazu angestellt, ob $\mathcal{DLR}_{reg,ifd}$ um Inklusionszusicherungen über reguläre Ausdrück erweitern werden kann, um somit zumindest eine interessante Teilmenge von semantischen Bedingungen modellieren zu können. Das Studium der Literatur zu *Description Logics* gibt jedoch berechtigten Grund zu der Vermutung, dass diese Erweiterung ebenfalls zu einer unentscheidbaren *Description Logic* führen würde.

Abstract

Object-oriented data models can be described by means of different formal methods syntactically and semantically. Among these formalisms are *Frame-Logic* and *Description Logics*. *Uniqueness Constraints* are an important class of semantic constraints that can be imposed on a database scheme. The meaning of a *Uniqueness Constraint* is, that certain values or objects are equal. Path functional dependencies (PFDs), functional dependencies (FDs) and keys belong to the class of *Uniqueness Constraints*. This diploma thesis tries to solve the question, whether the logical implication of semantic conditions, which also cover *Uniqueness Constraints*, is decidable within the formalism of the *Frame Logic*. The main question is whether the logical implication for PFDs, class inclusion constraints (CICs) and onto-constraints (OCs, which declare a kind of accessibility warranties) is decidable within the *Frame Logic*. In this thesis, path functional dependencies are in the focus.

For many *Description Logics*, the logical implication problem is known to be decidable. Therefor, to investigate the decidability of the logical implication problem in *Frame-Logic*, this problem is to be reformulated in *Description Logics*. The attempt is to find a reduction from the logical implication in *Frame-Logic* to the logical implication within a *Description Logic*.

The results show that the problem cannot be reduced to any of the examined *Description Logics* ($\mathcal{DLR}_{reg,ifd}$, `DLClass`, $\mathcal{DLFRD}$ and $\mathcal{ALCQI}_{reg}$), since the available constructs within each of these formalisms are insufficient to model all the three classes of semantic conditions considered. Methods to create new, more powerful formalisms like the union or fusion of the regarded *Description Logics* fail likewise, since the classes of semantic conditions to be modelled require the combination of incompatible constructs, resulting in an undecidable implication problem.

Finally, in order to model at least an interesting subset of semantic conditions, it is considered, whether inclusion constraints for regular expressions can be added to $\mathcal{DLR}_{reg,ifd}$. The study of the literature on *Description Logics* gives reason to the assumption that this extension will likewise lead to an undecidable *Description Logic*.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Objektorientierte Datenmodelle 1

1.2 Schema und Instanz objektorientierter Datenbanken 4

1.3 Semantische Bedingungen 4

1.4 Erfüllbarkeits- und Implikationsprobleme für semantische Bedingungen 8

1.5 Entscheidbarkeit und Komplexität 9

1.6 Motivation der Arbeit 11

1.7 Ziel der Arbeit 12

1.8 Vorgehensweise 12

2 Einführung in die F-Logic **14**

2.1 Beschreibung der F-Logic nach Biskup und Polle 14

2.2 Folgern in der F-Logic 20

3 Einführung in Description Logics **23**

3.1 Description Logics 23

3.2 Vorstellung der DL $\mathcal{DLR}_{reg,ifd}$ 31

3.3 Entscheidungsverfahren für $\mathcal{DLR}_{reg,ifd}$ 40

3.4 Vorstellung der DL `DLClass` 56

3.5 Vorstellung der DL $\mathcal{ALCQI}_{reg}$ 58

3.6 Semantische Bedingungen in DL 60

4 Stand der Forschung **66**

4.1 Relationales Datenmodell 66

4.2 F-Logic 67

4.3 Graphbasierte Datenmodelle 67

4.4 Description Logics 69

5 Ansätze zur Reduktion der F-Logic auf DL **75**

5.1 Prinzip: Modellierung der F-Logic in DL 75

5.2 Zu übertragende Konstrukte 76

5.3 Modellierung der F-Logic in $\mathcal{DLR}_{reg,ifd}$ 77

5.4 Modellierung der F-Logic in DLClass 93

5.5 Modellierung der F-Logic in $\mathcal{ALCQI}_{reg}$ 95

5.6 Verbindungen von DL 97

5.7 Erweiterung von $\mathcal{DLR}_{reg,ifd}$ 101

6 Zusammenfassung und Ausblick **108**

6.1 Ergebnisse 108

6.2 Ausblick 110

A Abkürzungsverzeichnis **112**

Tabellenverzeichnis **113**

Literaturverzeichnis **114**

Kapitel 1

Einleitung

In dieser Einleitung soll zunächst das objektorientierte Datenmodell als Ausgangspunkt für die Problemstellung illustriert werden. Dann werden die Begriffe des Datenbankschemas und der Instanz eingeführt. Anschließend werden semantische Bedingungen für Datenbankschemata vorgestellt und das Implikationsproblem für semantische Bedingungen in Datenbanken beschrieben. Nach einigen Bemerkungen zum Begriff der Entscheidbarkeit und Komplexität von Problemen wird zur Motivierung des Themas dessen Einordnung in den Kontext seiner möglichen Anwendungen im Bereich der objektorientierten Datenbanksysteme vorgenommen. Das Kapitel schließt mit der Formulierung des Ziels dieser Arbeit ihrer Gliederung.

1.1 Objektorientierte Datenmodelle

Objektorientierte Datenmodelle (OODMs) bilden die Struktur der Wirklichkeit ausgehend von Seienden (*Objekten*), deren Eigenschaften (*Attributen*), Verhalten (Verarbeiten von Nachrichten durch den Aufruf von *Methoden*) und Beziehungen zu anderen Objekten (*Struktur*) ab.

Objekte sind einmalig und während ihrer gesamten Lebensdauer innerhalb eines Datenbanksystems über ein eindeutiges Merkmal, das *Surrogat*, identifizierbar; während sich andere Eigenschaften eines Objektes verändern können, bleibt das Surrogat während der gesamten Lebensdauer des Objekts innerhalb der Datenbank invariant. Objekte können eine *innere Struktur* aufweisen. Dies bedeutet, dass man sich derartige Objekte als Container für andere Objekte vorstellen kann. Auf diese „enthaltenen“ Objekte wird innerhalb der Datenbank unter Verwendung der Surrogate *verwiesen*. Auf enthaltene Objekte kann mittels der Attribute des Objekts zugegriffen werden. Der Aufruf eines Attributs veranlasst das Datenbanksystem dann dazu, den entsprechenden Verweis durch das vorgefundene Surrogat auszuwerten und liefert das so identifizierte Objekt als „Wert“ des Attributs.

Eine Menge von Objekten, welche über die gleichen inneren Strukturen (Attribute) und gleichartiges Verhalten (Methoden) verfügen, kann man als Menge von *Instanzen* einer *Klasse* auffassen. Eine Klasse wird daher als Beschreibung von gemeinsamen Strukturen und Verhaltensweisen formal beschrieben (Klassenbeschreibung). Insbesondere wenn man bei der Betrachtung von Attributen und Methoden beschreiben will, für welche Klasse von Objekten sie definiert werden oder welche Klasse von Objekten sie gemäß Definition liefern sollen, fasst man eine Klasse auch als einen *Typ* von Objekten auf. Attribute und Methoden werden dann in einer Klassenbeschreibung durch *Signaturen getypt*, d.h. ihnen werden Typen für Domäne (Definitionsbereich) und Wertebereich (Bildbereich) zugewiesen.

Klassen oder Typen können in *Hierarchien* angeordnet werden. Die Hierarchie entspricht dabei einer Ordnung auf den Klassen bzw. Typen, die besagt, dass eine bestimmte Klasse C_1 eine *Oberklasse* einer anderen Klasse C_2 sei. Die Klasse C_2 nennt man dann auch *Unterklasse* von C_1. Die durch die Klassenhierarchie auszudrückende Beziehung entspricht der von Verfeinerung bzw. der Verallgemeinerung der Klassenbeschreibungen. Eine Oberklasse ist dann eine Verallgemeinerung all ihrer Unterklassen und jede Unterklasse ist eine Verfeinerung ihrer Oberklassen. Dies bedingt, dass Merkmale von einer Oberklasse an eine Unterklasse *vererbt* werden. Jede Unterklasse verfügt über sämtliche Attribute und Methoden ihrer Oberklassen, kann diese jedoch verfeinern (spezialisieren) oder ihnen zusätzliche Attribute oder Methoden hinzufügen. Jede Instanz einer Klasse C wird außerdem als Instanz sämtlicher Oberklassen von C aufgefasst.

Die Betrachtung des Attributs A eines Objekts o kann man als *Anwendung* einer Funktion A auf das Objekt betrachten. Der Wert des Attributs ist dann $A(o)$.

Ketten von durch Punkte verbundenen Attributen nennt man *Pfadbeschreibungen*.

Ebenso wie ein Attribut kann man auch eine Pfadbeschreibung Pf auf ein Objekt o anwenden. Pf(o) nennt man dann eine *Pfadfunktion*. Eine alternative und übliche Schreibweise zu Pf(o) ist o.Pf. Der Wert einer Pfadfunktion wird durch die iterierte Auswertung der Verweise, welche die einzelnen Attribute der Pfadbeschreibungen nacheinander liefern, ermittelt.

Für jedes Objekt sei außerdem prinzipiell das Attribut „Id“ definiert. Dieses Attribut (die *Identität*) verweist immer auf das Objekt selbst zurück.

Beispiel 1.1 (Pfade). Sei die Klasse „Person“ wie folgt beschrieben:

```
Person[  Name: String
         Alter: Zahl
         Vater: Person
         Mutter: Person]
```

Dies bedeutet, dass die Klasse „Person“ vier Attribute habe. Die Signatur dieser Attribute sieht vor, dass ihre Anwendung auf ein Objekt vom Typ „Person“ Objekte folgenden Typs liefert: Für die Attribute „Vater“ und „Mutter“ Objekte von

Typ „Person“, für das Attribut „Name“ ein Objekt vom Typ „String“ und für das Attribut „Alter“ ein Objekt vom Typ „Zahl“. Die Typen „String“ und „Zahl“ seien im Datenbanksystem vorhandene primitive Typen. Für jedes Objekt sei außerdem „Id“ definiert.

In diesem Beispiel sind

Pf1 =`Id.Vater.Vater`,

Pf2 =`Vater.Vater.Alter` und

Pf3 =`Vater.Name`

Pfadbeschreibungen. Seien nun folgende Objekte des Typs Person gegeben:

p1:Person[Name: „Salomo“, Alter: 12, Vater: p2, Mutter: p3],

p2:Person[Name: „David“, Alter: 53, Vater: p4, Mutter: `NULL`],

p3:Person[Name: „Batseba“, Alter: 36, Vater: `NULL`, Mutter: `NULL`],

p4:Person[Name: „Isai“, Alter: 89, Vater: `NULL`, Mutter: `NULL`].

Im dargestellten Beispiel führen einige Berechnungen der Werte der Pfadfunktionen zu folgenden Ergebnissen:

$$
\begin{aligned}
\text{p1.Pf1} &= \text{p1.Id.Vater.Vater} \\
&= \text{(Id(p1)).Vater.Vater} \\
&= \text{p1.Vater.Vater} \\
&= \text{(Vater(p1)).Vater} \\
&= \text{p2.Vater} \\
&= \text{(Vater(p2))} \\
&= \text{p4} \\
\text{p1.Pf2} &= \text{p1.Vater.Alter} \\
&= \text{(Vater(p1)).Vater.Alter} \\
&= \text{p2.Vater.Alter} \\
&= \text{(Vater(p2)).Alter} \\
&= \text{p4.Alter} \\
&= \text{(Alter(p4))} \\
&= 89 \\
\text{p1.Pf3} &= \text{(Vater(p1)).Name} \\
&= \text{p2.Name} \\
&= \text{(Name(p2))} \\
&= \text{„David“} \\
\text{p2.Pf3} &= \text{(Vater(p2)).Name} \\
&= \text{p4.Name} \\
&= \text{(Name(p4))} \\
&= \text{„Isai“}
\end{aligned}
$$

Für die Objekte p2, p3, p4 sind die Pfade Pf1 und Pf2, für p3 und p4 ist der Pfad Pf3 nicht definiert, d.h. während des Fortschreitens der Berechnung von Pf(o) stößt man auf den Wert `NULL`, der angibt, dass der Wert des entsprechenden Attributs nicht definiert ist.

Eine ausführliche und formale Charakterisierung der Prinzipien objektorientierter Datenmodelle und objektorientierter Datenbanken würde hier zu weit führen, findet sich jedoch in einschlägigen Lehrbüchern, etwa dem von Biskup [Bis95b].

Zur Spezifikation von Klassenbeschreibungen und Signaturen gibt es unterschiedliche formale Sprachen, insbesondere eignen sich dazu verschiedene Arten von *Description Logic-Sprachen* und *F-Logic*. Derartige Formalismen werden in den Kapiteln 3 bzw. 2.1 vorgestellt.

1.2 Schema und Instanz objektorientierter Datenbanken

Das *Schema* einer objektorientierten Datenbank definiert Klassen für Objekte, insbesondere legt es für jede Klasse einen Klassennamen fest und gibt Signaturen für sämtliche Attribute und Methoden jeder Klasse an. Außerdem legt das Schema eine Klassenhierarchie auf den definierten Klassen fest. Zusätzlich können weitere Bedingungen formuliert werden, die man *semantische Bedingungen* nennt (dazu mehr im folgenden Abschnitt 1.3).

Eine *Instanz* einer objektorientierten Datenbank ist eine Belegung der Datenbank mit Objekten, die dem Schema der Datenbank genügt. Dazu muss in der Regel jedes in der Datenbank vorhandene Objekt mindestens einer der im Schema beschriebenen Klassen angehören. Außerdem muss jedes Objekt über die Attribute aller Klassen verfügen, die es instantiiert. Dazu werden den Attributen des Objekts Werte oder auch Surrogate als Verweise zu anderen Objekten innerhalb der Datenbank zugeordnet. Diese Wertebelegung muss den Signaturen der jeweiligen Attribute entsprechen. Außerdem muss die Instanz die gegebenenfalls definierten semantischen Bedingungen des Schemas berücksichtigen.

1.3 Semantische Bedingungen

In Datenbanken geben *Semantische Bedingungen* (*semantic constraints*) Einschränkungen für Instanzen des definierten Datenbankschemas wieder.

In dieser Arbeit werden insbesondere folgende Klassen semantischer Bedingungen betrachtet:

(1) Uniqueness Constraints (UCs):

- Pfadabhängigkeiten (*path functional dependencies* – PFDs)
- Funktionale Abhängigkeiten (*functional dependencies* – FDs)
- Schlüsselbedingungen (*key constraints*)

(2) Onto-Abhängigkeiten (*onto constraints*)

(3) Klasseninklusionsabhängigkeiten (*subsumption constraints*)

Gerade diese Klassen semantischer Bedingungen wurden in einer Arbeit von Biskup und Polle [BP03] gemeinsam in einem objektorientierten Datenmodell berücksichtigt, von dem in dieser Arbeit untersucht werden soll, ob die logische Implikation innerhalb dieses Modells entscheidbar ist.

Im Folgenden sollen diese Arten von semantischen Bedingungen allgemein vorgestellt werden. Innerhalb der Beschreibungen der in dieser Arbeit angewandten Formalismen der *Description Logic* und der *F-Logic* werden spezielle Formalisierungen innerhalb dieser Sprachkonzepte vorgestellt.

Neben den in dieser Arbeit vorgestellten Klassen von semantischen Bedingungen gibt es noch weitere Abhängigkeiten, etwa mehrwertige (*multi valued dependencies* – MVDs) oder Verbundabhängigkeiten (*join dependencies* – JDs), die hier jedoch keine Beachtung finden werden (vgl. dazu etwa [Bis95b], Seiten 48, 98 f.).

Im Folgenden seien A, A', A_i, A'_i Attribute, C, C_i Klassen und $\mathrm{Pf}, \mathrm{Pf}', \mathrm{Pf}_i, \mathrm{Pf}'_i$ sogenannte *Pfade* über Attribute, also verkettete Aufrufe von Attributen von einem Objekt aus. Id bezeichnet die Identität.

1.3.1 Uniqueness Constraints

Ein *Uniqueness Constraint* ist eine semantische Bedingung, die besagt, dass unter vorgegebenen Bedingungen gewisse Werte oder Objekte in der Datenbank gleich sind. Uniqueness Constraints können z.B. dazu genutzt werden, um in objektorientierten Datenbanken Objekte über ihre Eigenschaften zu identifizeren (*weak value-identifiability/ value representability*, vgl. [ST93]). Spezialfälle von Uniqueness Constraints sind Pfadabhängigkeiten, funktionale Abhängigkeiten und Schlüsselbedingungen. Dabei gibt die genannte Reihenfolge auch eine Hierarchie der Typen von Uniqueness Constraints wieder: So sind Schlüsselbedingungen eine Verfeinerung der funktionalen Abhängigkeiten, und diese eine Verfeinerung von Pfadabhängigkeiten.

Zu einer weiteren Möglichkeit der Charakterisierung unterschiedlicher Klassen von Uniqueness Constraints siehe auch [WHB92].

1.3.1.1 Pfadabhängigkeiten

Eine *Pfadabhängigkeit* (*path functional dependency*, PFD) ist eine verallgemeinerte funktionale Abhängigkeit (siehe unten) für objektorientierte Datenmodelle. Anstatt auf einfache Attribute, greift eine Pfadabhängigkeit auf Pfadfunktionen zurück. Die Pfadabhängigkeit besagt, dass wenn man eine erste Menge von Pfadfunktionen auf zwei Objekte anwendet und dabei paarweise übereinstimmende Werte erhält, auch die Ergebnisse der Anwendung einer zweiten Menge von Pfadfunktionen paarweise miteinander übereinstimmen.

Seien $\mathrm{Pf}_1, \ldots, \mathrm{Pf}_n, \mathrm{Pf'}_1, \ldots, \mathrm{Pf'}_k$ Pfadbeschreibungen. Eine Datenbankinstanz genügt einer Pfadabhängigkeit

$$(\mathrm{Pf}_1, \ldots, \mathrm{Pf}_n \longrightarrow \mathrm{Pf}'_1, \ldots, \mathrm{Pf}'_k)$$

gdw. für alle Objekte o, o' der Datenbankinstanz gilt: Wenn die Pfade $\mathrm{Pf}_1, \ldots, \mathrm{Pf}_n$ für o, o' definiert sind, so sind dies auch die Pfade $\mathrm{Pf}'_1, \ldots, \mathrm{Pf}'_k$ und es gilt:

$$(\forall i \in \{1, \ldots, n\} : o.\mathrm{Pf}_i = o'.\mathrm{Pf}_i) \Rightarrow (\forall i \in \{1, \ldots, k\} : o.\mathrm{Pf'}_i = o'.\mathrm{Pf'}_i).$$

Offensichtlich kann jede funktionale Abhängigkeit durch eine äquivalente Pfadabhängigkeit ersetzt werden.

Die Pfadbeschreibungen $\mathrm{Pf}_1, \ldots, \mathrm{Pf}_n$ (auf der linken Seite des Pfeils) werden in dieser Arbeit *Prämissen*, die Pfadbeschreibungen $\mathrm{Pf}'_1, \ldots, \mathrm{Pf}'_k$ (rechts des Pfeils) dagegen *Konsequenzen* der Pfadabhängigkeit genannt.

1.3.1.2 Funktionale Abhängigkeiten

Funktionale Abhängigkeiten stammen aus dem relationalen Datenmodell. Eine *funktionale Abhängigkeit* (*functional dependency*, FD) $(A_1, \ldots, A_n \longrightarrow A'_1, \ldots, A'_k)$ fordert dabei, dass wenn auf zwei Objekten eine Menge von Attributen $\{A_1, \ldots, A_n\}$ definiert ist und die Objekte bezüglich der Werte dieser Attribute miteinander übereinstimmen, auch alle Elemente einer zweiten Menge von Attributen $\{A'_1, \ldots, A'_k\}$ für beide Objekte allesamt definiert sind und dass deren Werte für beide Objekte jeweils miteinander übereinstimmen müssen.

Seien $A_1, \ldots, A_n, A'_1, \ldots A'_k$ Attribute. Eine Datenbankinstanz genügt der funktionalen Abhängigkeit

$$(A_1, \ldots, A_n \longrightarrow A'_1, \ldots, A'_k)$$

gdw. für alle Objekte o, o' in der Datenbankinstanz, für die $A_1, \ldots, A_n$ definiert sind, $A'_1, \ldots, A'_k$ ebenfalls definiert sind und gilt

$$\forall i \in \{1, \ldots, n\} : A_i(o) = A_i(o')$$
$$\Rightarrow \forall i \in \{1, \ldots, k\} : A'_i(o) = A'_i(o').$$

1.3.1.3 Schlüsselbedingungen

Auch Schlüsselbedingungen entstammen dem relationalen Datenmodell. Eine *Schlüsselbedingung* (*key constraint/identification constraint*) verlangt, dass alle Objekte bezüglich einer bestimmten Menge von Attributen paarweise *unterschiedliche* Kombinationen von Werten aufweisen. Die Werte dieser Attribute können dann als ein Schlüssel oder ein identifizierendes Merkmal für das entsprechende Individuum betrachtet werden, d.h. jedes Objekt ist über die Kombination seiner Werte auf diesen Attributen eindeutig identifizierbar.

Seien $A_1, \ldots, A_n$ Attribute. Eine Datenbankinstanz genügt der Schlüsselbedingung

$$(\mathbf{key} : A_1, \ldots, A_n)$$

gdw. für alle Objekte o, o' der Datenbankinstanz gilt:
Sind die Attribute $A_1, \ldots, A_n$ definiert, so gilt:

$$(A_1(o) = A_1(o')) \wedge \ldots \wedge (A_n(o) = A_n(o'))) \Rightarrow o = o'.$$

Schlüsselbedingungen sind Spezialfälle von funktionalen Abhängigkeiten, nämlich solche funktionale Abhängigkeiten ($K \longrightarrow V$), in denen V gleich der Menge aller Attribute und $K \subseteq V$ ist.

1.3.2 Onto-Abhängigkeiten

Eine *Onto-Abhängigkeit* (*onto constraint*) $C_2\{A|C_1\}$ besagt, dass jedes Objekt o, das Instanz einer Klasse C_1 ist, gleichzeitig Wert eines Attributs A ist, das auf Objekten einer Klasse C_2 definiert ist.

Somit existiert in einer gültigen Datenbankinstanz zu jeder Instanz o der Klasse C_1 eine Instanz o' der Klasse C_2, so dass gilt:

$$A(o') = o.$$

Eine Onto-Abhängigkeit garantiert also die „Erreichbarkeit“ aller Objekte einer Klasse von Objekten einer zweiten Klasse aus.

Onto-Abhängigkeiten werden von Biskup und Polle in [BP00, BP01] eingeführt. Sie spielen dort eine wichtige Rolle im Zusammenhang mit der Entwicklung von Normalenformen für objektorientierte Schemata durch *Pivoting*. Dieses Transformation ermöglicht es, durch eine Vereinfachung des Schemas die Überprüfung von semantischen Bedingungen — und damit die Update-Operationen in objektorientierten Datenbanken — zu beschleunigen.

1.3.3 Inklusionsabhängigkeiten

Inklusionsabhängigkeiten (*inclusion dependencies*, INDs) besagen, dass eine bestimmte Menge von Objekten in einer bestimmten anderen Mengen von Objekten enthalten sein muss.

Im relationalen Datenmodell verwendet man zur formalen Beschreibung einer Inklusionsabhängigkeit Projektionen π_{M_1} und π_{M_2}. Dies sind Funktionen, die jeweils eine n-äre Relation R_1 bzw. eine k-äre Relation R_2 auf eine Teilmenge M_1 bzw. M_2 ihrer Komponenten (Attribute) abbilden. Die Mengen M_1, M_2 müssen dazu gleich mächtig sein.

Eine Inklusionsabhängigkeit $M_1(R_1) \subseteq M_2(R_2)$ fordert dann, dass Tupel, die als Instanzen einer durch die Projektion von R_1 auf die Teilmenge M_1 ihrer Attribute (Komponenten) gebildeten Relation vorkommen, gleichfalls Instanzen einer zweiten derartigen Relation derselben Stelligkeit sind, die durch die Projektion von R_2 auf deren Attributmenge M_2 entsteht:

$$\pi_{M_1}(R_1) \subseteq \pi_{M_2}(R_2)$$

In dieser Arbeit werden insbesondere *Klasseninklusionsabhängigkeiten* (*class subsumption constraints*, CICs) berücksichtigt. Als unäre Inklusionsabhängigkeiten sind dies die Spezialfälle, in denen Projektionen ausschließlich auf einelementige Attribut-/Komponentenmengen erlaubt sind. Eine Klasseninklusionsabhängigkeit besagt *letztlich*, dass alle Objekte, die Instanzen einer ersten Klasse sind, gleichfalls Instanzen einer zweiten Klasse sind.

Seien C_1, C_2 Klassen. Eine Datenbankinstanz genügt der Klasseninklusionsabhängigkeit

$$(C_1 \subseteq C_2)$$

gdw. für alle Objekte o gilt:

o ist Instanz von $C_1 \Rightarrow o$ ist Instanz von C_2.

Eine solche Klasseninklusionsabhängigkeit gilt etwa, falls C_1 ein Untertyp von C_2 ist.

1.4 Erfüllbarkeits- und Implikationsprobleme für semantische Bedingungen

Bei Untersuchungen des Schemaentwurfs, der Anfrageoptimierung und weiteren Anwendungen stellt sich die Frage danach, ob eine bestimmte semantische Bedingung

aus dem vorliegenden Datenbankschema logisch folgt (Folgerungsproblem). Äquivalent ist die Frage nach der Hülle der semantischen Bedingungen eines Datenbankschemas, also der Menge aller semantischen Bedingungen, die man aus dem vereinbarten Schema folgern kann.

Das Erfüllbarkeitsproblem befasst sich mit der Fragestellung, ob es zu einem gegebenen Datenkankschema eine nicht-triviale (also nicht leere) Instanz (eine Ausprägung des Schemas) geben kann, also danach, ob das Schema konsistent (d.h. in sich nicht widersprüchlich) ist. Verständlicherweise möchte man inkonsistente Schemata schon beim Entwurf vermeiden.

Das Implikationsproblem befasst sich mit der Frage, ob Implikationsbeziehungen zwischen semantischen Bedingungen in einem Datenbankschema algorithmisch erkannt werden können, und, falls dies bejaht werden kann, mit der Frage nach der Komplexität eines solchen Verfahrens.

Für viele bekannte Datenmodelle, etwa für die *F-Logic* nach Biskup und Polle und die meisten *Description Logics* wie die $\mathcal{DLR}$- und $\mathcal{ALC}$-Varianten, unterscheidet man zwischen dem *endlichen* und dem *generellen* oder *allgemeinen* Erfüllbarkeits- und Implikationsproblem. Während man im endlichen Fall danach fragt, ob es eine endliche Instanz, das heißt eine Instanz der Datenbank mit endlich vielen Objekten gibt, fragt das generelle Implikationsproblem danach, ob es überhaupt eine — vielleicht auch unendliche — Instanz des Datenbankschemas geben kann. Entsprechend fragt die endliche logische Implikation nach der Folgerbarkeit in endlichen Datenbanken, während das generelle Implikationsproblem nach der Folgerbarkeit in unbeschränkten Datenbanken fragt.

1.5 Entscheidbarkeit und Komplexität

Die Frage nach der *Entscheidbarkeit* von Implikationsproblemen zielt auf die Frage, ob ein Algorithmus angegeben werden kann, der bei Eingabe eines Datenbankschemas und einer semantischen Bedingung entscheidet, ob diese aus dem Datenbankschema logisch folgt. Ein Algorithmus ist ein schematisches, iterierendes Verfahren, das auf jede Eingabe (d.h. konkrete Fragestellung)

(1) nach endlich vielen Schritten terminiert und

(2) dann ein korrektes Ergebnis ausgibt.

Kann für ein Problem ein solcher Algorithmus angegeben werden, so ist das Problem *entscheidbar*. Kann bewiesen werden, dass kein solcher Algorithmus existiert, so ist das Problem *unentscheidbar*.

Ist ein Problem entscheidbar, so stellt sich in der Regel die Frage nach der *Komplexität* des Entscheidungsalgorithmus. Die Komplexität gibt dabei ein Maß für den

Aufwand an, den die Berechnung eines Algorithmus für ein Problem verursacht. Sie stellt somit ein Kriterium für die Beantwortung der Frage dar, ob das Problem mit einem vertretbaren Aufwand, also *effizient*, lösbar ist. Dabei geht man von drei Fällen aus, dem schlechtest möglichen Fall (*worst case*), dem bestmöglichen Fall (*best case*) und dem zu erwartenden Fall (*average case*).

Das Maß der Komplexität charakterisiert die Relation zwischen dem Bedarf an Zeit (Anzahl der Rechenschritte) oder Platz (benötigter Speicherplatz während des Ablaufs des Algorithmus'), und der Größe (Länge) der Eingabe. Es wird in Form einer mathematischen Funktion in der Eingabegröße angegeben.

Bemerkung 1.1 (Kodierung von Zahlen). Im Weiteren gelte stets die im Bereich der *Description Logics* und *Propositional Dynamic Logics* gängige Konvention, dass sämtliche *Zahlen* in Ein- und Ausgaben in unärer Darstellungsweise kodiert werden. Komplexitätsmaße beziehen sich, soweit nicht ausdrücklich Anderweitiges angegeben wird, immer auf derart kodierte Ein-/Ausgaben.

Über den Begriff der *Komplexitätsklasse* werden unterschiedliche Probleme in Klassen zusammengefasst, die bezüglich ihrer algorithmischen Komplexität vergleichbar „schwer" sind, deren Lösung im *worst case* also vergleichbaren Aufwand erfordert.

Für diese Arbeit sind folgende Komplexitätsklassen von Bedeutung, deren Definition von der Berechnungskraft unterschiedlicher formaler Maschinen abhängig gemacht wird:

- PTIME – *polynomial time*: Klasse aller Probleme, für die es eine deterministischen Turingmaschine gibt, die alle Instanzen des Entscheidungsproblems in polynomieller Zeit (bezogen auf die Größe der Eingabe) löst.
- EXPTIME bzw. DEXPTIME – *deterministic exponential time*: Klasse aller Probleme, für die es eine deterministische Turingmaschine gibt, die alle Instanzen des Entscheidungsproblems in exponentieller Zeit (bezogen auf die Größe der Eingabe) löst.
- PSPACE – *polynomial space*: Klasse aller Probleme, für die es eine (nicht-) deterministische Turingmaschine gibt, die alle Instanzen des Entscheidungsproblems löst und dabei höchstens polynomiell viele (bezogen auf die Größe der Eingabe) unterschiedliche Speicherzellen auf dem Band liest.
- NEXPTIME – *nondeterministic exponential time*: Klasse aller Probleme, für die es eine nichtdeterministische Turingmaschine gibt, die alle Instanzen des Entscheidungsproblems in exponentieller Zeit (bezogen auf die Größe der Eingabe) löst.

Entscheidungsvariante nennt man die Formulierung eines Problems, für die man auf eine Eingabe mit „akzeptiert" oder „nicht akzeptiert" antworten kann. Als *Sprache*

L_i bezeichnet man diejenigen Worte über einem Alphabet Σ_i, die eine Turingmaschine T_i als Eingabe akzeptiert.

Sind L_1 und L_2 Sprachen über Eingabealphabete Σ_1 und Σ_2 für zwei Turingmaschinen T_1 und T_2, so dass T_i die Sprache $L_i \subseteq \Sigma_i^*$ erkennt, so bedeutet $L_1 \leq_p L_2$, dass es eine Funktion $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ gibt, die Eingaben für T_1 auf Eingaben für T_2 abbildet. Dabei hat f eine höchstens polynomielle Zeitkomplexität und es gilt $\alpha \in L_1 \Leftrightarrow f(\alpha) \in L_2$. Man sagt, „$L_1$ ist polynomiell auf L_2 reduzierbar“. Anschaulich formuliert bedeutet $L_1 \leq_p L_2$ soviel wie „L_1 ist algorithmisch höchstens so komplex wie L_2“

Für eine Komplexitätsklasse K heißt eine Sprache L

- *K-hart* gdw. für alle $L' \in K$ gilt: $L' \leq_p L$.
- *K-vollständig* gdw. $L \in K$ und für alle $L' \in K$ gilt: $L' \leq_p L$.

Die Probleme der Erfüllbarkeit und der logischen Implikation in formalen Logiken lassen sich leicht als Entscheidungsvarianten formulieren. Dazu definiert man für das Problem der Erfüllbarkeit eines Datenbankschemas eine Sprache L_1, die aus allen Datenbankschemata besteht, die erfüllbar sind. Für die logische Implikation definiert man eine Sprache L_2, so dass L_2 gerade aus sämtlichen Kombinationen von einem Datenbankschema und einer semantischen Bedingung besteht, so dass die semantische Bedingung logisch aus dem Schema folgt.

1.6 Motivation der Arbeit

Effiziente Algorithmen zur Inferenz von semantischen Bedingungen sind für viele wesentliche Aufgaben im Bereich des Designs von Datenbanken erforderlich.

Während für den Bereich des relationalen Datenmodells zahlreiche Ergebnisse zur Entscheidbarkeit und Komplexität der Inferenz vorliegen, ist dies für objektorientierte Datenmodelle bislang nicht der Fall.

Im objektorientierten Datenmodell finden Inferenz-Algorithmen beispielsweise folgende Anwendungen [Bis95a]:

- Durch *Pivoting* kann der Versuch unternommen werden, die Komplexität von Update-Operationen (Überprüfen des Einhaltens der semantischen Bedingungen) zu reduzieren. Das Pivoting muss jedoch über semantische Bedingungen gesteuert werden, anhand derer entschieden werden kann, ob sich die Transformation lohnt oder nicht [BP00, BP01].

- Das Ziel von *Sharing* ist es, Redundanzen innerhalb der Datenbank zu vermeiden. Dazu kann man im Datenbankschema mittels Hüllenbildung bezüglich der Implikation über die semantischen Bedingungen „verbotene Strukturen“ identifizieren und anschließend das Schema algorithmisch in ein äquivalentes Schema transformieren, das der Entwurfsheuristik „Trennung der Aspekte“ folgt.

- Zur Unterstützung von *Sichten* (*views*) in objektorientierten Datenbanksystemen bedarf es ebenfalls eines Verfahrens zur automatischen Inferenz, etwa um die Äquivalenz zweier Schemata auf der syntaktischen Ebene zu überprüfen.

- Die Behandlung von Klasseninklusionsabhängigkeiten mit anderen semantischen Bedingungen erlaubt die Untersuchung von Typen und Klassen sowie Klassenhierarchien in objektorientierten Datenmodellen.

Eine umfangreiche Sammlung von möglichen Anwendungen und Fragestellungen findet sich bei Biskup [Bis95a].

1.7 Ziel der Arbeit

In einer Arbeit von Biskup und Polle [BP03] wurde eine *F-Logic* vorgestellt, in der Klasseninklusionsabhängigkeiten, Pfadabhängigkeiten und Onto-Abhängigkeiten formuliert werden können. Für das Implikationsproblem wurde ein vollständiger und korrekter Ableitungskalkül entwickelt. Offen blieb jedoch die Frage nach der Entscheidbarkeit des Implikationsproblems.

In dieser Arbeit soll versucht werden, diese offene Frage zu beantworten. Dazu soll untersucht werden, ob sich bekannte *Description Logics* eignen, das Implikationsproblem für die Klasse von Klasseninklusionsabhängigkeiten, Pfadabhängigkeiten und Onto-Abhängigkeiten darzustellen und mittels bekannter Entscheidungsverfahren für diese Formalismen zu lösen.

1.8 Vorgehensweise

In diesem ersten Kapitel wurde die Fragestellung dieser Arbeit vorgestellt und motiviert.

Im folgenden, zweiten Kapitel werden die *F-Logic* und der Ableitungskalkül aus [BP03] dargestellt.

Anschließend werden im dritten Kapitel nach einer allgemeinen Einführung in *Description Logics* Syntax und Semantik verschiedener *Description Logic*-Sprachen

vorgestellt. Für die *Description Logic*-Sprache $\mathcal{DLR}_{reg,ifd}$ wird auch das Entscheidungsverfahren in seiner Gesamtheit vorgestellt.

Das vierte Kapitel bietet eine Übersicht über bekannte Forschungsergebnisse mit Bezug zur zu behandelnden Fragestellung. Insbesondere werden Ergebnisse zu Pfadabhängigkeiten und zu verschiedenen *Description Logic*-Varianten vorgestellt.

Im fünften Kapitel werden, unter Betrachtung der im vierten Kapitel eingeführten *Description Logic*-Sprachen, Lösungsansätze für die anstehende Fragestellung entwickelt und diskutiert.

Das letzte, sechste Kapitel schließt die Arbeit mit einer Zusammenfassung der Ergebnisse und einem Ausblick ab.

Kapitel 2

Einführung in die F-Logic

Dieses Kapitel dient der Vorstellung der von Biskup und Polle in [BP03] untersuchten Variante der *F-Logic*. Die Definitionen und Sätze wurden dem genannten Aufsatz entnommen.

2.1 Beschreibung der F-Logic nach Biskup und Polle

Biskup und Polle führen in [BP03] ein Datenmodell ein, das auf der von Kifer et al. in [KLW95] eingeführtem *Frame-Logic*, kurz: *F-Logic*, beruht. Das modifizierte Modell ist streng getypt, lässt jedoch mengenwertige Attribute außer Betracht. Im Folgenden wird die Variante aus [BP03] vorgestellt.

ID-Terme (Bezeichner) werden benutzt, um Objekte, Klassen und Methoden zu bezeichnen. $\mathcal{F}$ sei eine Menge von ID-Termen für Konstanten, $\mathcal{V}$ eine Menge von ID-Termen für Variablen. Der Ausdruck $C :: D$ ist eine *class is-a assertion*, d.h. C ist eine Unterklasse der Klasse D. Der Ausdruck $O : C$ ist eine *object is-a assertion*, d.h. das durch O repräsentierte Objekt ist Mitglied der Klasse C.

Objekt-Moleküle haben die Form O[Liste von Methodenausdrücken], wobei ein Methodenausdruck entweder die Form $A \rightarrow V$ (skalarer Datenausdruck: der Aufruf der Methode A für Objekt O liefert den Wert V) oder $A \Rightarrow R$ (skalarer Signaturausdruck, Methode A liefert für Objekte der Klasse O Werte der Ergebnisklasse R zurück) hat.

Formeln sind molekulare Formeln oder durch Konnektoren oder Quantoren verknüpfte Formeln. Ist X eine Variable und sind ϕ und ψ Formeln, so sind ebenfalls Formeln: $\phi \wedge \psi, \phi \vee \psi, \neg\phi, \exists X\phi, \forall X\phi$. $\phi \longleftarrow \psi$ sei eine Schreibweise für $\phi \vee \neg\psi$. Konstrukte, die keine Variable enthalten, heißen Grund- oder variablenfreie Terme. Formeln, die keine ungebundene Variable enthalten, heißen *Aussagen*.

$U(\mathcal{F})$, das *Herbrand-Universum*, ist die Menge aller Grundterme. Die Menge von variablenfreien Molekülen $\mathcal{HB}(\mathcal{F})$ heißt *Herbrand-Basis*. Eine unter logischer Implikation abgeschlossene Teilmenge von $\mathcal{HB}(\mathcal{F})$ heißt *H-Struktur*. Eine H-Struktur heißt *H-Modell* einer Formel, wenn diese Formel logisch aus der H-Struktur folgt.

Definition 2.1 (Schema). *Ein Schema D hat die Form*

$$D = \langle \underbrace{CL_D \cup AT_D \cup HR_D \cup SG_D}_{ST_D} | SC_D \rangle$$

mit den folgenden, endlichen Mengen:

- *Menge von „Konstanten“ $CL_D \subset \{t[] | t \in \mathcal{F}\}$ für* Klassennamen*,*
- *Menge von „Konstanten“ $AT_D \subset \{t[] | t \in \mathcal{F}\}$ für* Attributsnamen*,*
- *Menge von variablenfreien class is-a assertions $HR_D \subset \{c :: d | c[], d[] \in CL_D\}$, welche die (azyklische)* Klassenhierarchie *bilden,*
- *eine Menge von variablenfreien Molekülen, die ausschließlich aus skalaren Signaturausdrücken besteht, $SG_D \subset \{c[a \Rightarrow r] | c[], r[] \in CL_D, a[] \in AT_D\}$, und die* Signaturen *für Klassen und Attribute deklariert,*
- *einer Menge von Aussagen, SC_D, die als* semantische Bedingungen *die Menge der erlaubten Instanzen einschränkt. In unserem Fall beschränken wir diese Menge auf $SC_D = CIC_D \cup OC_D \cup PFD_D$ (Definitionen siehe unten).*

Es gelte ferner $CL_D \cap AT_D = \emptyset$. Das Schema sei zeitinvariant.

Definition 2.2 (Axiome). *Folgende Axiome erzwingen bestimmte Eigenschaften für Instanzen des Schemas:*

- unique-name axioms $UN = \{\overset{\circ}{\neq}(t, t') | t, t' \in U(\mathcal{F}), t \not\equiv t'\}$
  (D.h. für jeden ID-Grundterm gibt es höchstens einen Bezeichner. Dies schließt unter anderem zyklische Deklarationen wie $\{C :: D, D :: C\} \subset HR_D$ aus.)
- not-null axiom $NN = \{\forall C \forall A \forall O \exists V : \quad O[A \rightarrow V] \longleftarrow C[A \Rightarrow ()] \wedge O : C\}$
  (D.h. wenn für eine Klasse eine Methodensignatur deklariert wurde, liefert diese Methode für Objekte dieser Klasse immer Werte zurück.)
- well-typedness axioms
  $WT = \{\forall O \forall A \forall V \exists C : \quad (C[A \Rightarrow ()] \wedge O : C) \longleftarrow O[A \rightarrow V]\}$
  $\cup \{\forall O \forall A \forall V \forall C \forall R : \quad V : R \longleftarrow C[A \Rightarrow R] \wedge O : C[A \rightarrow V]\}$
  (D.h. wenn eine Methode einen Wert liefert, dann ist auch eine Signatur deklariert worden, und wenn eine Methode auf einem Objekt einen Wert liefert und eine Methodensignatur für diese Klasse deklariert wurde, so hat der gelieferte Wert einen der Signatur entsprechenden Ergebnistyp.)

Schließlich sei die Menge der Axiome definiert durch $AX := UN \cup NN \cup WT$.

Definition 2.3 (Eigentliche Attribute). *Sei* D *ein Schema.*

- $At_D(c) := \{a | HR_D \cup SG_D \models c[a \Rightarrow ()]\}$ *heißt* Menge der Attribute der Klasse c.
- *Ein Attribut* $a \in AT_D$ *heißt* eigentliches Attribut einer Klasse c *gdw.* $SG_D \models c[a \Rightarrow ()]$ *und* $SG_D \not\models d[a \Rightarrow ()]$ *gilt für alle Klassen* $d \in CL_D$ *mit* $d \not\equiv c$.

Demnach ist ein eigentliches Attribut nur für genau eine Klasse definiert, die auch keine Unterklassen haben darf. Nach [BP03, Anmerkung zu Definition 4] ist das logische Folgern über *is-a assertions* und skalare Signaturausdrücke für endliche Mengen von *is-a assertions* und skalare Signaturausdrücke entscheidbar. HR_D und SG_D können also zu einer vollständigen Klassenhierarchie und einer vollständigen Signaturenmenge erweitert werden.

Definition 2.4 (Extension). *Eine* Extension $f \in ext(D)$ *eines Schemas* D *hat die Gestalt* $f = \langle pp_f | ob_f \rangle$ *mit*

- *einer Menge von variablenfreien object is-a assertions, welche die Klassen bevölkern („Population“):* $pp_f \subset \{o{:}c | o \in \mathcal{F} \setminus (CL_D \cup AT_D), c \in CL_D\}$,
- *einer Menge von variablenfreien Objektmolekülen mit ausschließlich skalaren Datenausdrücken, welche die Attributwerte auf den Objekten festlegen:* $ob_f \subset \{o[a \rightarrow v] | o, v \in \mathcal{F} \setminus (CL_D \cup AT_D), a \in AT_D\}$.

Definition 2.5 (Abschluss). *Sei* f *eine Extension des Schemas* D. *Der* Abschluss *der Extension* f *unter dem Schema* D *ist*

$$compl_D(f) := \{\tau \in \mathcal{HB}(\mathcal{F}) | ST_D \cup pp_f \cup ob_f \models \tau\}.$$

Abschlüsse sind die kleinsten H-Modelle der Vereinigung $ST_D \cup pp_f \cup ob_f$.

Definition 2.6 (Instanz). *Sei* f *eine Extension zum Schema* D. f *heißt* Instanz *des Schemas* D, $f \in sat(D)$, *wenn* $compl_D(f)$ *ein H-Modell der Axiome* AX *und der semantischen Bedingungen* SC_D *ist.*

Definition 2.7 (Objekt- und Klassenmarkierungen). *Sei* f *eine Extension zum Schema* D.

- *Die Menge aller Objekte in der Extension* f *wird notiert als* $obj(f) := \{o | ex.\ c :\ o{:}c \in pp_f\}$.

- *Die Klassenmarkierung* $\lambda_{Cl}(o)$ *für ein Objekt* $o \in obj(f)$ *ist die Menge der Klassen, deren Mitglied o ist:* $\lambda_{Cl}(o) := \{c | HR_D \cup pp_f \models o:c\}$.

Im Folgenden werden die einzelnen Klassen von Abhängigkeiten vorgestellt. Für jeden Typ wird zuerst dessen „informelle“ Notation und im Anschluss daran eine Abhängigkeitsformel vorgestellt, vermittels derer die Abhängigkeit in SC_D formal dargestellt wird. Außerdem wird zu jedem Abhängigkeitstyp eine Formelmenge definiert, welche genau alle Abhängigkeitsformeln des entsprechenden Typs enthält.

Definition 2.8 (Klasseninklusionsabhängigkeit – CIC).

- *Eine* Klasseninklusionsabhängigkeit *für Klassen* $c, d \in CL_D$ *über einem Schema D hat die Form* $c \subset d$.
- *Die* Klasseninklusionsabhängigkeitsformel *für eine Klasseninklusionsabhängigkeit* $c \subset d$ *ist* $\forall O : \quad O:d \longleftarrow O:c$.
- CIC_D *bezeichne die Menge aller Klasseninklusionsabhängigkeitsformeln in* SC_D.

Definition 2.9 (Onto-Abhängigkeit – OC).

- *Eine* Onto-Abhängigkeit *für eine Klasse* $c \in CL_D$ *über einem Schema D hat die Form* $c\{a|d\}$, *wobei* $a \in At_D(c)$ *ein eigentliches Attribut von c ist und* $d \in CL_D$ *eine Klasse.*[1]
- *Die* Onto-Abhängigkeitsformel *für eine Onto-Abhängigkeit* $c\{a|d\}$ *ist*

$$\forall V \exists O : \quad (O:c[a \to V] \wedge c[a \Rightarrow ()]) \longleftarrow V:d.$$

- *Für eine Klasse c bezeichne* $OC_D(c)$ *die Menge der Onto-Abhängigkeitsformeln für c.* OC_D *bezeichne die Menge aller Onto-Abhängigkeitsformeln in* SC_D.

Definition 2.10 (Abschluss für CICs und OCs).

- *Sei* $\vdash_D$ *der Folgerungsoperator gemäß Definition 12 in [BP03].*
- *Die Menge aller CICs und OCs über einem Schema D wird wie folgt definiert:* $\Upsilon_D := CIC_D \cup OC_D$.
- *Der* Abschluss *der Menge* Υ_D, *notiert* $\Upsilon_D^{+_D}$, *ist die Menge aller Klasseninklusionsabhängigkeiten und Onto-Abhängigkeiten* ν *über dem Schema D, so dass gilt* $\Upsilon_D \vdash_D \nu$.

[1] Die Beschränkung der OCs auf eigentliche Attribute ergibt sich nach [BP03, Def. 37 und Anmerkungen] aufgrund von Wechselwirkungen der OC mit der Vererbung von Signaturen und PFDs.

Definition 2.11 (Folder). *Der* Folder *einer Klasse $c \in CL_D$ ist definiert durch* $Folder_D(c) := \{d | \Upsilon_D \vdash_D \ c \subset d\}$.

Der Folder eine Klasse c umfasst alle Klassen, denen ein Objekt aus Klasse c gemäß den semantischen Bedingungen und der Klassenhierarchie angehören muss.

Definition 2.12 (Beliebige Wegbeschreibungen).

- *Die Menge der* beliebigen Wegbeschreibungen $W_{arb}(D)$ *über dem Schema D besteht aus allen endlichen Folgen von durch Punkte oder Minuszeichen getrennten Attributnamen und aus der* Identischen Wegbeschreibung *.Id (*$.Id \notin \mathcal{V} \cup \mathcal{F}$*).*

$$\begin{aligned} W_{arb}(D) &:= \{\pi_1 m_1 \cdots \pi_n m_n | n > 0 \text{ und } m_i \in AT_D, \pi_i \in \{.,-\}, i \in \{1,\ldots,n\}\} \\ &\cup \{.Id\} \end{aligned}$$

- *Die Länge der Wegbeschreibung .Id ist 0, len(.Id):=0. Die Länge einer Wegbeschreibung* $w \equiv \pi_1 m_1 \cdots \pi_n m_n$ *ist* n, $len(w) = n$.
- *Die Menge der* beliebigen Pfadfunktionen $P_{arb}(D) \subset W_{arb}(D)$ *über einem Schema D besteht aus allen beliebigen Wegbeschreibungen, in denen ausschließlich Punkte auftreten.*

Definition 2.13 (Verkettung beliebiger Wegbeschreibungen).
Seien $w_1, w_2 \in W_{arb}(D)$ *beliebige Wegbeschreibungen über Schema D. Die Verkettung beider,* $w_1 \circ w_2$, *wird wie folgt definiert:*

$$w_1 \circ w_2 := \begin{cases} .Id & \text{falls } w_1 = w_2 = .Id \text{ oder } w_1 = \pi m, w_2 = \bar{\pi} m \\ w_1 & \text{falls } w_1 \neq .Id, w_2 = .Id \\ w_2 & \text{falls } w_1 = .Id, w_2 \neq .Id \\ w_1' & \text{falls } w_1 = w_1' \pi m, w_2 = \bar{\pi} m \\ w_2' & \text{falls } w_1 = \pi m, w_2 = \bar{\pi} m w_2' \\ w_1' \circ w_2' & \text{falls } w_1 = w_1' \pi m, w_2 = \bar{\pi} m w_2' \\ w_1 w_2 & \text{sonst} \end{cases}$$

$$\text{wobei } \bar{\pi} := \begin{cases} . & \text{falls } \pi = - \\ - & \text{falls } \pi = . \end{cases}$$

Definition 2.14 (Wohlgeformte Wegbeschreibungen).

- *Die Menge der wohlgeformten Wegbeschreibungen* $W_{wf}(D) \subset W_{arb}(D)$ *über einem Schema D ist die kleinste Menge, so dass*

– $.Id \in W_{wf}(D)$, *wobei*

$$Dom_D(.Id) := CL_D \text{ und}$$
$$Ran_D(c, .Id) := Folder_D(c) \text{ für alle } c \in CL_D,$$

– *wenn* $w \in W_{wf}(D)$ *eine wohlgeformte Wegbeschreibung mit einer Domänenklasse* $c \in Dom_D(w)$ *ist, so dass* $a \in \bigcup_{d \in Ran_D(c,w)} At_D(d)$ *für ein beliebiges Attribut* a *und* $len(w) < len(w \circ .a)$, *dann gilt* $w \circ .a \in W_{wf}(D)$, *wobei*

$$\begin{aligned} Dom_D(w \circ .a) &:= \{e \in Dom_D(w) | a \in \bigcup_{f \in Ran_D(e,w)} At_D(f)\} \text{ und} \\ Ran_D(e, w \circ .a) &:= \{h \in Folder_D(g) | \text{ex. } f : f \in Ran_D(e,w) \text{ und} \\ &\quad HR_D \cup SG_D \models f[a \Rightarrow g]\} \\ &\quad \text{für alle } e \in Dom_D(w \circ .a), \end{aligned}$$

– *wenn* $w \in W_{wf}(D)$ *eine wohlgeformte Wegbeschreibung mit einer Domänenklasse* $c \in Dom_D(w)$ *ist, so dass* $d\{a|b\} \in \Upsilon_D^{+_D}$ *für eine Klasse* $b \in Ran_D(c, w)$ *und* $len(w) < len(w \circ -a)$, *so gilt* $w \circ -a \in W_{wf}(D)$, *wobei*

$$\begin{aligned} Dom_D(w \circ -a) &:= \{e \in Dom_D(w) | \text{ex. } f, g : f \in Ran_D(e,w) \text{ und} \\ &\quad g\{a|f\} \in \Upsilon_D^{+_D}\} \text{ und} \\ Ran_D(e, w \circ -a) &:= \{h \in Folder_D(g) | \text{ex. } f : f \in Ran_D(e,w) \text{ und} \\ &\quad g\{a|f\} \in \Upsilon_D^{+_D}\} \\ &\quad \text{für alle } e \in Dom_D(w \circ -a). \end{aligned}$$

- *Für jede Klasse* $c \in CL_D$ *bezeichnet* $WayDes_D(c)$ *die Menge aller wohlgeformten Wegbeschreibungen* $w \in W_{wf}(D)$, *wobei* $c \in Dom_D(w)$; *das sind alle wohlgeformten Wegbeschreibungen, die in der Klasse* c *starten.*

- *Die Menge* $PathFuncs_D(c) \subset WayDes_D(c)$ *bezeichnet die Menge aller wohlgeformten Pfadfunktionen, die in der Klasse* c *starten.*

$\mathrm{Dom}_D(w)$ ist die Menge der Klassen, in deren Objekten die wohlgeformte Wegbeschreibung w starten kann, während $\mathrm{Ran}_D(e, w)$ die Menge derjenigen Klassen angibt, in denen Objekte liegen können, die man von Objekten einer Klasse e aus erreichen kann, wenn man die wohlgeformte Wegbeschreibung w anwendet (siehe folgende Definition 2.15).

Definition 2.15 (Anwendung von Wegbeschreibungen). *Sei* f *Instanz des Schemas* D, $o \in obj(f)$ *ein Objekt,* $c \in \lambda_{Cl}(o)$ *eine Klasse und* $w \in WayDes_D(c)$ *eine Wegbeschreibung.*

- *Falls* $w = .Id$, *dann* $o.w = \{o\}$.
- *Falls* $w = w' \circ .a$, *dann* $o.w = \{o'' | ex.\ o' :\ o' \in o.w'$ *und* $ob_f \models o'[a \rightarrow o'']\}$.
- *Falls* $w = w' \circ -a$, *dann* $o.w = \{o'' | ex.\ o' :\ o' \in o.w'$ *und* $ob_f \models o''[a \rightarrow o']\}$.

Definition 2.16 (Pfadabhängigkeit – PFD).

- *Eine* Pfadabhängigkeit *für eine Klasse* $c \in CL_D$ *über einem Schema* D *hat die Form* $c(p_1 \cdots p_k \rightarrow p_{k+1} \cdots p_n)$, *wobei* $k \in \{1, \ldots, n-1\}$ *und* $p_i \in PathFuncs_D(c)$ *für* $i \in \{1, \ldots, n\}$.
- *Die* PFD-Formel *für eine Pfadabhängigkeit* $c(p_1 \cdots p_k \rightarrow p_{k+1} \cdots p_n)$ *lautet*

$$X[p \rightarrow P] \quad \longleftarrow \quad \begin{array}{l} X : c[p_1 \rightarrow P_1; \ldots; p_k \rightarrow P_k] \wedge \\ Y : c[p_1 \rightarrow P_1; \ldots; p_k \rightarrow P_k; p \rightarrow P] \end{array}$$

  für alle $p \in \{p_{k+1}, \ldots, p_n\}$. *Dabei sei* $X[p \rightarrow Y]$ *mit*

$$p \equiv .m_1^p \cdots .m_l^p \in P_{arb}(D) \setminus \{.Id\}$$

  eine Kurzschreibweise für

$$(\exists X_1^p \cdots \exists X_{l-1}^p : \quad (X[m_1^p \rightarrow X_1^p] \wedge X_1^p[m_2^p \rightarrow X_2^p] \wedge \cdots \wedge X_{l-1}^p[m_l^p \rightarrow Y]))$$

  und $X[.Id \rightarrow Y]$ *seine eine Schreibweise für* $X \stackrel{\circ}{=} Y$.
- PFD_D *bezeichne die Menge aller PFD-Formeln zu Pfadabhängigkeiten in* SC_D.

2.2 Folgern in der F-Logic

In [BP03] wird ein System von 12 Inferenzregeln für das Folgern in dem vorgestellten Dialekt der F-Logic vorgestellt sowie dessen Vollständigkeit und Korrektheit bewiesen. In diesem Abschnitt werden lediglich die Inferenzregeln und die Hauptsätze vorgestellt. Deren Beweise sind der genannten Arbeit zu entnehmen.

Definition 2.17 (Inferenzregeln). *Sei* π *eine PFD über dem Schema* D. *Die PFD* π *ist* herleitbar *aus der Menge* Ξ_D, *notiert* $\Xi_D \vdash_D \pi$, *gdw.* $\pi \in \Xi_D$ *oder* π *ist aufgrund einer oder mehrerer der folgenden 12 Inferenzregeln herleitbar:*

I1 – Transitivität der Klasseninklusion
Sind $c \subset c'$ *und* $c' \subset c''$ *herleitbar sind, so ist auch* $c \subset c''$ *herleitbar.*

I2 – Inklusion von Unterklassen
Wenn $HR_D \models c :: c'$ *gilt, so ist* $c \subset c'$ *herleitbar.*

I3 – Inklusion der Signatur
Wenn $d\{a|c\}$ herleitbar ist, wobei $HR_D \cup SG_D \models d[a \Rightarrow c']$, so ist $c \subset c'$ herleitbar.

I4 – Restriktion des Wertebereichs
Wenn $c \subset c'$ und $d\{a|c'\}$ herleitbar sind, so auch $d\{a|c\}$.

I5 – Restriktion des Definitionsbereichs
Wenn $c \subset c'$ und $c\{a|d\}$ herleitbar sind, wobei $a \in At_D(c')$, so kann $c'\{a|d\}$ hergeleitet werden.

I6 – Reflexivität der PFDs
Für jede Klasse $c \in CL_D$ und alle nicht-leeren Mengen $Y \subset X$ von Pfadfunktionen, für die X eine endliche Teilmenge von $PathFuncs_D(c)$, ist $c(X \rightarrow Y)$ herleitbar.

I7 – Augmentierung von PFDs
Für jede Klasse $c \in CL_D$ und eine endliche Teilmenge Z von $PathFuncs_D(c)$ gilt: Wenn $c(X \rightarrow Y)$ herleitbar ist, so auch $c(XZ \rightarrow YZ)$ (wobei XZ (YZ) die Vereinigung aller Pfadfunktionen in X (Y) und Z bezeichne).

I8 – Transitivität von PFDs
Sind sowohl $c(X \rightarrow Y)$ als auch $c(Y \rightarrow Z)$ herleitbar,
so ist auch $c(X \rightarrow Z)$ herleitbar.

I9 – Einfache Attribution
Für jede Klasse $c \in CL_D$ und jedes Attribut $a \in At_D(c)$ ist $c(.Id \rightarrow .a)$ herleitbar.

I10 – Einfache Präfixaugmentierung
Für jede Klasse $c_1 \in CL_D$ und jedes Attribut $a \in At_D(c_1)$:
Ist $c_2(p_1 \cdots p_k \rightarrow p_{k+1} \cdots p_n)$ herleitbar, wobei $c_2 \in Ran_D(c_1, .a)$,
so ist $c_1(.a \circ p_1 \cdots .a \circ p_k \rightarrow .a \circ p_{k+1} \cdots .a \circ p_n)$ herleitbar.

I11 – Einfache Präfixreduktion
Für jede Klasse $c_1 \in CL_D$, für die $c_2(.a \circ p_1 \cdots .a \circ p_k \rightarrow .a \circ p_{k+1} \cdots .a \circ p_n)$ herleitbar ist, wobei $c_2\{a|c_1\} \in \Upsilon_D^{+_D}$, ist $c_1(p_1 \cdots p_k \rightarrow p_{k+1} \cdots p_n)$ herleitbar.

I12 – Erblichkeit von PFDs
Für jede Klasse $c_1 \in CL_D$: Wenn $c_2(X \rightarrow Y)$ herleitbar ist, wobei $c_1 \subset c_2 \in \Upsilon_D^{+_D}$, so ist $c_1(X \rightarrow Y)$ herleitbar.

Definition 2.18 (Abschluss für CICs, OCs und PFDs). *Der* Abschluss $\Xi_D^{+_D}$ *der Menge Ξ_D ist die Menge aller semantischen Bedingungen ξ über dem Schema D, für die gilt: $\Xi_D \vdash_D \xi$.*

Satz 2.1 (Korrektheit der Inferenzregeln). *Die Inferenzregeln I1 bis I12 sind korrekt, das heißt: Für die Menge Ξ_D der semantischen Bedingungen in einem Schema D und eine semantische Bedingung ξ über dem Schema D gilt: Wenn $\xi \in \Xi_D^{+_D}$ eine semantische Bedingung ist, dann erfüllt jede Instanz des Schemas D die semantische Bedingung ξ: $sat(D) \subset sat(D \cup \{\xi\})$.*

Satz 2.2 (Vollständigkeit der Inferenzregeln I1 – I5). *Sei D ein Schema und ν eine CIC oder OC über dem Schema D, so dass υ nicht aus der Menge Ξ_D der semantischen Bedingungen im Schema D hergeleitet werden kann (d.h. $\Xi_D \nvdash_D \upsilon$). Dann existiert eine Instanz f des Schemas D, so dass f keine Instanz von $D \cup \{\upsilon\}$ ist.*

Satz 2.3 (Vollständigkeit der Inferenzregeln I6 – I12). *Sei D ein Schema und $c(X \rightarrow Y) \notin \Xi_D^{+_D}$ PFD über dem Schema D. Dann existiert eine Instanz f des Schemas D, die keine Instanz von $D \cup \{c(X \rightarrow Y)\}$ ist.*

Abschließend sei das Problem der logischen Implikation von semantischen Bedingungen in der *F-Logic* angesprochen.

Definition 2.19 (Logische Implikation in der F-Logic). *Eine semantische Bedingung γ folgt logisch aus einem Schema D, $D \models \gamma$, genau dann wenn gilt: Jede Instanz von D ist auch eine Instanz von $D \cup \{\gamma\}$.*

Bemerkung 2.1 (Benennung von Elementen in SC_D, CIC_D, OC_D und PFD_D). Für den Rest dieser Arbeit werden zur Verbesserung der Lesbarkeit für die Elemente der Mengen SC_D, CIC_D, OC_D und PFD_D Abkürzungen eingeführt. Anstelle der jeweiligen Abhängigkeitsformel tritt dann die Abhängigkeit selbst, also etwa $(c \subset d)$ an Stelle von $\forall V \exists O: \ (O : c[a \rightarrow V] \wedge c[a \Rightarrow ()]) \longleftarrow V : d$.

Kapitel 3

Einführung in Description Logics

In diesem Kapitel werden *Description Logics* (DLs) vorgestellt. Nach einem kurzen geschichtlichen Abriss der Entwicklung von *Description Logics* werden die grundlegenden Prinzipien von *Description Logics* sowie die Unterschiede zwischen verschiedenen *Description Logic*-Sprachen informal vorgestellt. Anschließend werden drei *Description Logic*-Sprachen eingeführt, die in Kapitel 5 Verwendung finden werden. Dies sind die Sprachen $\mathcal{DLR}_{reg,ifd}$, `DLClass` und $\mathcal{ALCQI}_{reg}$. Die erstgenannte Sprache ist eine Kombination der *Description Logic*-Sprachen $\mathcal{DLR}_{ifd}$ [CGL01] und $\mathcal{DLR}_{reg}$ [CGL98]. Sowohl für diese beiden Sprachen an sich, als auch für beider Kombination ist das Problem der logischen Implikation auf Wissensbasen entscheidbar und EXPTIME-vollständig [CGL01].

3.1 Description Logics

3.1.1 Geschichtliche Entwicklung der DL

Die Wiedergabe der geschichtlichen Entwicklung der *Description Logics* folgt der Darstellung bei *Nebel und Smolka* [NeS91]. Nach dieser Darstellung richten sich auch die Literaturbelege innerhalb dieses Abschnitts. Eine weitere übersichtliche Darstellung bieten Nardi und Brachmann in [NB03].

Description Logics entstanden aus zwei Wurzeln:

(1) *terminological logics*, die zur Repräsentation von Konzepten und Terminologien in Anwendungen der Künstlichen Intelligenz entwickelt wurden,

(2) *feature logics*, die aus der Computerlinguistik stammen und dort zur Repräsentation semantischer und syntaktischer Information in Sätzen natürlicher Sprachen dienen.

Terminological logics basieren auf dem Formalismus KL-ONE (Brachman, [Bra79]). Sie definieren ausgehend von primitiven Konzepten und Rollen (mengenwertigen Attributen) neue Konzepte. Zu den üblichen Leistungen von terminologischen Repräsentationssystemen gehört die Berechnung der Konzepthierarchie, basierend auf Inklusionsbeziehungen zwischen den Konzepten, und die Berechnung der Instanzbeziehung zwischen Konzepten und Objekten.

Feature logics basieren auf Unifikationsgrammatiken (*Lexical Functional Grammar* – LFG, Kaplan und Bresman, [KM82]; *Functional Unification Grammar* – (FUG), Kay, [Kay79]). Syntaktische und semantische Objekte werden attributiv mittels sogenannter *features* beschrieben. Features sind dabei einwertige Attribute. *Feature paths* sind Kompositionen solcher funktionaler Attribute. Die syntaktische und semantische Struktur von natürlichsprachlichen Sätzen werden so durch Unifikation miteinander in Übereinstimmung gebracht. Typisch ist die Überprüfung einer neuen Beschreibung auf Erfüllbarkeit. Das Konzeptenthaltenseinsproblem und Unerfüllbarkeitsproblem sind gegenseitig linearzeit-reduzibel [NeS90, NeS91].

Beiden Formalismen liegt eine Tarski-Modelltheorie zu Grunde, allerdings unterscheiden sich die angewandten Algorithmen aufgrund der unterschiedlichen Auffassung von Attributen bezüglich ihrer Komplexität beträchtlich.

Über Korrespondenzen von *terminological logics* zu nichtdeterministischen endlichen Automaten konnten Erkenntnisse zur Entscheidbarkeit und Komplexität von *terminological logics* sowohl mit azyklischen, als auch mit zyklischen Terminologien erzielt werden. Das Konzeptinklusionsproblem für *terminological logics* und *feature logic*s ist demnach co-NP-vollständig. Terminologische Zyklen können mittels einer Größter-/Kleinster-Fixpunkt-Semantik analysiert werden. Die Zurückführung auf die Automatentheorie zeigt dann, dass das Konzeptinklusionsproblem für diese Fälle PSPACE-vollständig ist [Neb90].

Erweiterungen von *terminological logics* um Gleichheitsbedingungen über *feature paths* führen zur Unentscheidbarkeit des Konzeptenhaltenseinsproblems. Als die entscheidende Voraussetzung für die Entscheidbarkeit wurden hierbei die funktionale Charakterisierung von Attributen erkannt. Ein Beweis der Unentscheidbarkeit basiert auf einer Korrespondenz zu unentscheidbaren Wortproblemen für umkehrbare Thue-Systeme. Mengenwertige Attribute in *feature paths* führen somit zur Unentscheidbarkeit (Schmidt-Schauss [Sch89]). Selbst die Beschränkung von *feature paths* auf funktionale Attribute, führt in Verbindung mit zyklischen Terminologien zur Unentscheidbarkeit des Enthaltenseinsproblems für deskriptive und Größte-Fixpunkt-Semantiken [Neb91].

Korrespondenzen von *terminological logics* und *propositional dynamic logics* sowie *modal dynamic logics* wurden erstmals von Schild aufgezeigt [Scd91]. Dies führte zu einer Vielzahl interessanter Ergebnisse (Existenz endlicher Modelle, Algorithmen, verschiedene Ergebnisse zur Komplexität). Konzeptenthaltenseinsprobleme für *terminological logics* können seit dem auf Entscheidbarkeitsprobleme in einer Modallogik reduziert werden, für die Entscheidbarkeit und Komplexität bekannt sind.

Problematisch bleiben hierbei jedoch *feature paths*, die keine direkte Entsprechung in modalen oder dynamischen Logiken finden.

3.1.2 Informale Beschreibung von DL-Sprachen

3.1.2.1 Grundlegende Prinzipien

Mittels *Description Logic*-Sprachen können Schemata von (objektorientierten) Datenbanken formal beschrieben werden. Objekte werden in diesen Formalismen *Individuen* genannt. Dazu bedient man sich einer Beschreibung durch *Rollen* (binäre Relationen über Individuen) und *Konzepte* (Mengen von Individuen). Vermittels sogenannter *Konstruktoren* werden aus primitiven (atomaren) Rollen- und Konzepten allgemeine Rollen- und Konzeptausdrücke aufgebaut. Diese allgemeinen Ausdrücke werden benutzt, um *Zusicherungen* (*assertions*) über Rollen, Konzepte und Individuen zu deklarieren. Zusicherungen in Hinblick auf die intensionale Beschreibung (Beschreibung von Rollen und Konzepten ohne Bezugnahme auf einzelne Individuen) werden der sogenannten *Terminologie* oder *TBox* zugerechnet. Insbesondere werden hier Inklusionszusicherungen über Konzepte und Rollen definiert. Zusicherungen bezüglich einer extensionalen Beschreibung (Beschreibungen, die einzelne Individuen oder Tupel betreffen, also die Extension von Konzepten und Relationen) bilden die sogenannte *ABox*. Hierbei handelt es sich z.B. um Instanz-Zusicherungen, die garantieren sollen, dass Konzepte oder Rollen nicht leer sind. Ein Schema $K = \langle \mathcal{T}_K | \mathcal{A}_K \rangle$ mit nicht-leerer TBox $\mathcal{T}_K$ und nicht-leerer ABox $\mathcal{A}_K$ wird *Wissensbasis* genannt.

Eine TBox bzw. eine Wissensbasis wird durch eine *Interpretation* instantiiert. Eine Interpretation besteht aus einer Interpretationsdomäne (einer Menge von Individuen) und einer Interpretationsfunktion, die jeder primitiven Rolle und jedem Konzept eine Tupelmenge bzw. eine Menge von Individuen der Interpretationsdomäne zuweist. Die *Semantik* der DL gibt insbesondere an, wie die Interpretationsfunktion auf den allgemeinen Rollen- und Konzeptausdrücken fortgesetzt wird.

Eine Interpretation ist ein *Modell* der TBox (bzw. der Wissensbasis), wenn alle in der TBox (Wissensbasis) definierten Zusicherungen gemäß der angewandten Semantik erfüllt werden.

3.1.2.2 Unterschiede zwischen Description Logic-Sprachen

Description Logic-Sprachen unterscheiden sich in Bezug auf ihre Ausdrucksstärke und — dadurch bedingt — ihre Entscheidbarkeit. Dabei entstehen die Differenzen aufgrund der Wahl

(1) der Menge der Konzeptkonstruktoren, Beispiele: universale/existentielle Quantifikation, Anzahlbeschränkungen;

(2) der Menge der Rollenkonstruktoren, Beispiele: inverse und transitive Rollen, Rollenverkettung, Bool'sche Konstruktoren, n-äre Relationen;

(3) der Art der Zusicherungen in der TBox (*TBox assertions*), Beispiele: CICs, RICs, Transitivität und Funktionalität von Rollen, *key-assertions*;

(4) der Art der Zusicherungen in der ABox (*ABox assertions*), Beispiele: Instanz-Zusicherungen;

(5) der Festlegung einer Semantik (deskriptive, Kleinster-/Größter-Fixpunkt-Semantik).

3.1.2.2.1 Konzeptkonstruktoren Folgende Liste führt alle wesentlichen Konstruktoren auf, mittels derer in *Description Logic*-Sprachen Konzeptausdrücke gebildet werden können. Die Konstruktoren kommen gelegentlich in abweichender Notation vor. C, C_i stehen für allgemeine Konzeptausdrücke, R, R_i für allgemeine Rollenausdrücke, $\mathbf{R_i}$ für Mengen von Rollenausdrücken, n für eine nicht-negative ganze Zahl. In runden Klammern folgt jeweils eine informale Beschreibung der üblichen Interpretation.

- $C_1 \sqcup C_2$, Konzeptvereinigung/Konzeptdisjunktion
  (Vereinigung beider Konzepte)
- $\neg C$, Komplementbildung
  (Interpretationsdomäne ohne Instanzen des Konzepts)
- $C_1 \sqcap C_2$, Konzeptdurchschnitt/Konzeptkonjunktion
  (Schnittmenge beider Konzepte)
- $\forall R.C$, universale Quantifikation/Werterestriktion (alle Individuen, bei denen die angegebene Rolle ausschließlich auf Instanzen des genannten Konzepts verweist)
- $\exists R.\top$, $[\exists R.C]$, [qualifizierte] existentielle Quantifikation (alle Individuen, bei denen die angegebene Rolle wenigstens auf eine Instanz des genannten Konzepts verweist)
- $(\leq nR)$, $[(\leq nR.C)]$, [qualifizierte] Minimum-Anzahlbeschränkung, R ist i.d.R. auf primitive Rollen und deren Inverse beschränkt (alle Individuen, bei denen die Rolle auf mindestens n Individuen des genannten Konzepts verweist)
- $(\geq nR)$, $[(\geq nR.C)]$, [qualifizierte] Maximum-Anzahlbeschränkung, R ist i.d.R. auf primitive Rollen und deren Inverse beschränkt (alle Individuen, bei denen die Rolle auf höchstens n Individuen des genannten Konzepts verweist)
- $\top$, TOP - das allgemeinste Konzept (entspricht der Interpretationsdomäne)

- $\bot$, BOTTOM, das leere Konzept (entspricht der leeren Menge)
- $(\mathbf{R_1} \sim \mathbf{R_2}), \sim \in \{\subseteq, =\}$, RVM – *role-value-map* (alle Individuen o, für die gilt: $\{o' \in \Delta^{\mathcal{I}} | (o, o') \in R_1^{\mathcal{I}}\} \sim \{o' \in \Delta^{\mathcal{I}} | (o, o') \in R_2^{\mathcal{I}}\}$)
- $(\mathbf{R_1} \downarrow \mathbf{R_2})$ $[(\mathbf{R_1} \not\downarrow \mathbf{R_2})]$, *feature-agreement* [*-disagreement*] für Ketten funktionaler Rollen $\mathbf{R_1}, \mathbf{R_2}$, auch *path-equation* oder `SAME-AS` genannt (Menge der Individuen, die über die Funktionsketten $\mathbf{R_1}, \mathbf{R_2}$ [nicht] mit denselben Individuum verbunden sind)
- $\{a\}; \{a_1, \ldots, a_n\}$, einelementiges Konzept, `FILLS`; `ONE-OF` (ein Konzept, das genau die aufgeführten Individuen umfasst)

3.1.2.2.2 Rollenkonstruktoren Den Konstruktoren zur Erzeugung von Konzeptausdrücken entsprechend gibt es auch Konstruktoren zur Bildung von Rollen- oder Relationsausdrücken. Auch hier existieren abweichende Notationsformen.

- $R_1 \sqcup R_2$, Rollenvereinigung
- $\neg R$, Rollenkomplement (eigentlich die Differenz der Universalrelation bezüglich der Interpretationsdomäne und der Rolle R, i.d.R. jedoch die Differenz von TOPROLE (s.u.) und der Rolle R)
- $R_1 \sqcap R_2$, Rollendurchschnitt
- R^-, Inverse Rolle
- $R_1 \circ R_2$, Verkettung (Komposition) von Rollen
- R^*, Reflexiv-transitiver Abschluss der Rolle R
- $R|_{[i][j]}$, Projektion einer n-ären Relation auf zwei ihrer Komponenten
- $\mathrm{Id}(C)$, Identität auf Instanzen des Konzepts C
- $\top_n$, TOPROLE (Vereinigung aller Rollen/Relation der Stelligkeit n)

3.1.2.2.3 TBox-Zusicherungen TBox-Zusicherungen sind Inklusionszusicherungen für Konzepte und — für den Fall, dass die *Description Logic* Rollenhierarchien erlaubt — Rollen:

- $C_1 \sqsubseteq C_2$, Konzeptinklusionszusicherung (CICs)
- $R_1 \sqsubseteq R_2$, Rolleninklusionszusicherung (RICs)

Nach der Form der TBox-Zusicherungen unterscheidet man mehrere Arten von Zusicherungen: Sei A ein Konzeptbezeichner bzw. ein primitives Konzept, C, C_1, C_2 seien beliebige Konzepte. Man unterscheidet dann folgende vier Arten von TBox-Zusicherungen:

(1) $A \sqsubseteq C$ (Primitive Konzeptbeschreibung)

(2) $A \equiv C$ (Konzeptdefinition)

(3) $C_1 \sqsubseteq C_2$ (Konzeptinklusion)

(4) $C_1 \equiv C_2$ (Konzeptgleichheit)

TBoxes, die nur TBox-Zusicherungen der Form (1) enthalten, nennt man *primitive TBoxes*. *Einfache TBoxes* können zusätzlich Zusicherungen der Form (2) enthalten. Sogenannte *freie TBoxes* enthalten außerdem Zusicherungen der Formen (3) oder (4) (vgl. ausführlichere Darstellung in [DLNS96]).

Weiterhin unterscheidet man zwischen *Description Logic*-Sprachen, die *zyklische Beschreibungen in der TBox* zulassen, und solchen, die derartige Terminologien ausschließen. Ein Zyklus in der TBox liegt dann vor, wenn ein Konzept, das auf der rechten Seite einer Zusicherung steht, direkt oder indirekt (durch andere Zusicherungen) auf dasselbe atomare Konzept verweist, das auf der linken Seite der Zusicherung erscheint.

Durch die Beschränkung der TBox $\mathcal{T}_K$ auf primitive Konzeptbeschreibungen kann man das Folgern über die Gleichheit von generalisierten Konzepten, wie etwa im Falle

$$\{(C_1 \sqsubseteq C_2), (C_2 \sqsubseteq C_1)\} \subseteq \mathcal{T}_K$$

vermeiden, das in vielen *Description Logics* zur Unentscheidbarkeit führt.

Man kann jedoch in Konzeptinklusionszusicherungen der Form $(C_1 \sqsubseteq C_2)$ die linken Seiten ggf. auf primitive Konzepte beschränken, ohne dass dies eine wirkliche Einschränkung der Problemklasse für die logische Inferenz darstellt, denn es gilt:

$$K \models (C_1 \sqsubseteq C_2)$$

kann unter Einführung eines neuen primitiven Konzepts C in K durch

$$K \cup \{(C \sqsubseteq C_1)\} \models (C \sqsubseteq C_2)$$

ersetzt werden (vgl. [KTW01]).

3.1.2.2.4 ABox-Zusicherungen Einige *Description Logics* erlauben Zusicherungen für die Zugehörigkeit von Individuen oder Tupel zu Konzepten bzw. Rollen:

- $C(a)$, Instanz-Zusicherung für ein Konzept
  (Individuum a ist im Konzept C enthalten)
- $R(t)$, Instanz-Zusicherung für eine Rolle
  (das Tupel t ist in der Rolle R enthalten)
- $x = y$, *Objektidentitätszusicherung*
  (die mit x und y bezeichneten Individuen sind identisch)
- $x \neq y$ (die mit x und y bezeichneten Individuen sind nicht identisch)

3.1.2.2.5 Semantiken Lässt die Syntax einer *Description Logic*-Sprache zyklische TBoxes zu, so muss eine Semantik für zyklische Ausdrücke festgelegt werden. Betrachtet man eine zyklische TBox-Zusicherung als eine Gleichung $A \equiv F(A)$, so gilt (vgl. [DLNS96]):

(1) *Deklarative Semantik*: In diesem Falle ist $A \equiv F(A)$ eine Zusicherung, die besagt, dass A *irgendeine* Lösung der entsprechenden Gleichung sein muss. Dann gilt allgemein: $A \equiv C$ ist äquivalent mit $\{A \sqsubseteq C, C \sqsubseteq A\}$. Diese Semantik erlaubt die Beschreibung notwendiger und hinreichender Bedingungen für Konzepte.

(2) *Kleinster-Fixpunkt-Semantik*: Die Gleichung verlangt dann, dass A die *kleinste* Lösung der Gleichung ist, falls eine solche existiert. Die Kleinster-Fixpunkt-Semantik eignet sich insbesondere für die induktive Beschreibung von Konzepten.

(3) *Größter-Fixpunkt-Semantik*: A wird hierbei als *größte* Lösung der Gleichung betrachtet, falls eine solche existiert. Diese Semantik eignet sich insbesondere zur Beschreibung von Klassen mit Individuen, deren Strukturen sich zirkulär wiederholen.

In den meisten *Description Logic*-Sprachen mit zyklischen TBoxes wird die deklarative Semantik angewandt.

3.1.2.3 Folgerungsprobleme in Description Logics

Sei K eine Wissensbasis. Das logische Folgern in *Description Logic* beschäftigt sich insbesondere mit den folgenden Fragestellungen:

(1) Folgern ohne TBox und ohne ABox ($K = \emptyset|\emptyset\rangle$)

(a) Konzepterfüllbarkeit (*satisfiability of concepts*, *concept consistency*, *concept satisfiability*): Ist ein Konzeptausdruck C ohne Rücksichtnahme auf weitere Konzeptausdrücke erfüllbar, d.h. gibt es ein Modell $\mathcal{I}$ von K mit $C^{\mathcal{I}} \neq \emptyset$, bzw. gilt $K \cup C \not\models \bot$?

(2) Folgern mit TBox, aber ohne ABox ($K = \langle \text{TBox} | \emptyset \rangle$)

(a) Konzepterfüllbarkeit (*concept satisfiability w.r.t. a TBox*): Gibt es ein Modell $\mathcal{I}$ von K, so dass $C^{\mathcal{I}} \neq \emptyset$ gilt bzw. $K \not\models (C \sqsubseteq \bot)$?

(b) TBox-Erfüllbarkeit (*satisfiability of TBoxes*, *TBox consistency*): Ist eine TBox konsistent, d.h. gibt es ein Modell für K bzw. gilt $K \not\models \bot$?

(c) TBox-Enthaltenseinsproblem (*logical implication in TBoxes*, *subsumption*): Gilt für alle Modelle von K, dass ein Konzept C_1 allgemeiner ist als ein zweites C_2, also $K \models (C_2 \sqsubseteq C_1)$?

(3) Folgern mit TBox und ABox ($K = \langle \text{TBox} | \text{ABox} \rangle$)

(a) Konzepterfüllbarkeit (*concept satisfiability w.r.t. a knowledge base K*): Gibt es ein Modell $\mathcal{I}$ von K, so dass $C^{\mathcal{I}} \neq \emptyset$ gilt, bzw. $K \not\models (C \sqsubseteq \bot)$?

(b) Enthaltenseinsproblem (*subsumption*): Ist ein Konzept C in einem anderen Konzept D enthalten, d.h. gilt für jedes Modell $\mathcal{I}$ von K, dass $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $K \models (C \sqsubseteq D)$?

(c) Wissensbasiserfüllbarkeit (*knowledge base satisfiability/consistency*): Ist eine Wissensbasis K erfüllbar, d.h. gibt es mindestens ein Modell von K, d.h. $K \not\models \bot$?

(d) Instanzüberprüfung (*instance checking*): Ist die Zusicherung $C(a)$ in jedem Modell von K erfüllt, $K \models C(a)$?

In unter den Bool'schen Operatoren über Konzepte (Konzeptkonstruktoren $\neg, \sqcap, \sqcup$) abgeschlossenen *Description Logic*-Sprachen mit deskriptiver Semantik lassen sich die Probleme (2a) und (2c), (3a) und (3b) sowie (3c) und (3d) aufeinander reduzieren: $K \models C \sqsubseteq D$ gdw. $K \models (C \sqcap \neg D \equiv \bot)$ und $K \models C(a)$ gdw. $K \cup \{\neg C(a)\} \models \bot$. Über das Prinzip der *Internalisierung der TBox* (das heißt die Reduktion einer TBox auf eine einzige Konzeptbeschreibung, vgl. [CGNL99]) können diese Probleme für viele *Description Logic*-Sprachen auf das Problem (1a), (2a) bzw. (3a) reduziert werden vgl. auch [DLNS94, Abschnitt 2.3]).

3.1.2.4 Entscheidungsprozeduren für Description Logics

Um Nutzen aus formalen, logischen Sprachen ziehen zu können, bedarf es eines Verfahrens, mittels dessen man sicher entscheiden kann, ob eine Formel die logische Konsequenz einer Formelmenge ist oder nicht. Ein solcher Algorithmus kann dann zum automatischen Beweisen genutzt werden. Auf diese Weise kann etwa die

Konsistenz eines Datenbankschemas überprüft werden, sofern es in der entscheidbaren Logik-Sprache formuliert wurde. Für diese Verfahren gibt es bei *Description Logic*-Sprachen zwei wesentliche Ansätze.

3.1.2.4.1 Tableau-Kalküle Entscheidungsprozeduren für DL beruhen häufig auf Widerlegungsbeweisen nach dem Tableau-Prinzip. Die Kontraposition der Formel, deren Erfüllbarkeit gezeigt werden soll, wird durch möglichst geschickte Auswahl und Anwendung von Inferenzregeln in äquivalente Formeln überführt, wobei das Ziel darin besteht, letztendlich eine offensichtlich unerfüllbare Formel abzuleiten. Dieser Widerspruch erweist dann die Erfüllbarkeit der ursprünglichen Formel. Diese Methode setzt eine vollständige und korrekte Axiomatisierung der Implikation voraus.

Beispiele: Tableau-Kalkül für $\mathcal{SHIF}$ in [HST99], Tableau-Algorithmus für $\mathcal{RIQ}$, das ist eine Erweiterung von $\mathcal{ALC}$ um Anzahlbeschränkungen, inverse Rollen, transitive Rollen und RICs [HS03].

3.1.2.4.2 Reduktion auf PDL Eine weitere Gruppe von *Description Logic*-Sprachen nutzt eine Kette polynomieller Reduktionen auf einfachere *Description Logic*-Sprachen und schließlich auf ein Entscheidbarkeitsproblem für eine äquivalente *propositional dynamic logic* (PDL). PDLs bilden eine Familie von modalen Logiken, die für die Untersuchung des Verhaltens von Programmen entwickelt wurden. Für diese Logiken sind Entscheidungsprozeduren bekannt. Als Beispiel für diese Art von Entscheidungsverfahren für *Description Logics* sei hier auf Abschnitt 3.3 verwiesen, in dem eine derartige Entscheidungsprozedur für die logische Implikation in der *Description Logic*-Sprache $\mathcal{DLR}_{reg,ifd}$ vorgestellt wird.

3.1.3 Ergebnisse für die Komplexität des Folgerns in unterschiedlichen DL-Sprachen

An dieser Stelle wird darauf verzichtet, eine Übersicht über die Komplexität und Entscheidbarkeit des Folgerns in unterschiedlichen *Description Logics* zu erstellen. Eine ausführliche und übersichtliche Darstellung findet sich bei Donini im Anhang zu [Don03].

3.2 Vorstellung der DL $\mathcal{DLR}_{reg,ifd}$

Die anschließende Vorstellung von $\mathcal{DLR}_{reg,ifd}$ orientiert sich im Wesentlichen an den Ausführungen von Calvanese et al. in [CGL01].

In $\mathcal{DLR}_{reg,ifd}$ wird eine Wissensbasis mit Hilfe formaler Beschreibungen definiert, die einer festgelegten Syntax genügen. Diese formalen Beschreibungen bilden das

Schema der Wissensbasis. Mittels geeigneter Interpretationen, die diesem Schema genügen, erhält man Modelle der Wissensbasis.

Dem Thema der Arbeit entsprechend wird fortan nur noch von statischen Datenbanken ausgegangen, das heißt Schema und Interpretation (Definitionen siehe unten) seien unveränderlich.

3.2.1 Syntax von $\mathcal{DLR}_{reg,ifd}$

Für die formale Beschreibung von Syntax und Semantik einer Wissensbasis werden Bezeichner benötigt. Diese werden aus einer Menge von Konstanten gewählt:

Definition 3.1 (Bezeichner). *Zur Identifizierung von Konstrukten der* Description Logic*, wie Konzepten, Relationen und Individuen, benötigt man eindeutige* Bezeichner *(*identifiers*). Für diese können beliebige Symbole aus einer fixen Symbolmenge $\mathcal{F}$ gewählt werden.*

In $\mathcal{DLR}_{reg,ifd}$ wird die Syntax einer Wissensbasis durch ein Schema beschrieben.

Definition 3.2 (Schema). *Das* Schema *(*scheme*) einer Wissensbasis wird folgendermaßen angegeben:*

$$D = \langle \underbrace{\mathcal{C}_D, \mathcal{R}_D}_{E_D} | \mathcal{T}_D | \mathcal{A}_D | \mathcal{S}_D \rangle.$$

Dabei bezeichnet

- $E_D \subseteq \mathcal{F}$ *eine endliche Menge von „Konstanten“, die Menge der* Bezeichner *in D, bestehend aus*
  - $\mathcal{C}_D$*, der Menge der* Konzeptbezeichner*,*
  - $\mathcal{R}_D$*, der Menge der* Relationsbezeichner*,*

  wobei die Mengen $\mathcal{C}_D$ und $\mathcal{R}_D$ disjunkt sind.
- $\mathcal{T}_D$ *die* Terminologie *oder* TBox*, das ist die Beschreibung von Konzepten und Relationen, in Gestalt einer endlichen Menge von* Konzept- *und* Relationsinklusionszusicherungen*, sowie Zusicherungen von semantischen Bedingungen für Konzepte und Relationen:*
  - $C_1 \sqsubseteq C_2$, [1]
  - $R_1 \sqsubseteq R_2$. [2]

[1] $C_1 \equiv C_2$ sei die Abkürzung für $C_1 \sqsubseteq C_2$, $C_2 \sqsubseteq C_1$.
[2] $R_1 \equiv R_2$ sei die Abkürzung für $R_1 \sqsubseteq R_2$, $R_2 \sqsubseteq R_1$.

Dabei sind C_1 und C_2 beliebige Konzeptausdrücke (siehe Def. 3.3 u. 3.6), R_1, R_2 beliebige Relationsausdrücke gleicher Stelligkeit arity(R_1) = arity(R_2) (siehe Definitionen 3.4 u. 3.6).[3]

- $\mathcal{A}_D$ *die* ABox*, eine endliche Menge von* ABox-Zusicherungen *(ABox assertions), die für Modelle der Wissensbasis gelten müssen. Die ABox-Zusicherungen sind von der Form*
  - $C(x)$, [4]
  - $R(x_1, \ldots, x_n)$,[5]
  - $x = y$ *oder*
  - $x \neq y$.

  wobei $x,\ x_1, \ldots, x_n, y \in \mathcal{F}_D \setminus (\mathcal{C}_D \cup \mathcal{R}_D)$ Skolemkonstanten, $R \in \mathcal{R}_D$, $n =$ arity(R) und $C \in \mathcal{C}_D$ sind.

- $\mathcal{S}_\mathcal{D}$ *eine Menge von ID- und FD-Zusicherungen der Formen*
  - $(\textbf{\textit{id}}\ C\ [i_1]R_1, \ldots, [i_h]R_h)$, $i_j \leq$ arity(R_j), *für* $1 \leq j \leq h$, *oder*
  - $(\textbf{\textit{fd}}\ R\, i_1, \ldots, i_h \rightarrow i)$, $h \geq 2; h, j \leq$ arity(R).

  Dabei ist C ein beliebiger Konzeptausdruck (siehe Definitionen 3.3 u. 3.6) und $R, R_1, \ldots, R_h$ sind beliebige Relationsausdrücke (siehe Definitionen 3.4 u. 3.6). ID- und FD-Zusicherungen formulieren bestimmte semantische Bedingungen (siehe Abschnitte 1.3 und 3.6; vgl. [CGL01]).

Zu einem Schema $K = \langle E_K | \mathcal{T}_K | \mathcal{A}_K | \mathcal{S}_K \rangle$ und einer TBox T bzw. einer ABox A — wobei in T und A ausschließlich solche primitive Relationen und Konzepte auftreten, die bereits in K bekannt sind — heiße

$$\begin{aligned} K \cup T &:= \langle E_K | \mathcal{T}_K \cup T | \mathcal{A}_K | \mathcal{S}_K \rangle \\ K \cup A &:= \langle E_K | \mathcal{T}_K | \mathcal{A}_K \cup A | \mathcal{S}_K \rangle \end{aligned}$$

die Erweiterung von K um T bzw. um A.

Während die TBox universell quantifizierte Zusicherungen enthält (Zusicherungen für Konzepte), beziehen sich die ABox-Zusicherungen auf individuelle Objekte. Somit kann man bestimmte Eigenschaften für Konzepte (Def. 3.3), Relationen (Def.

[3] Nach [GL96] stellen Zyklen in der Terminologie kein Problem für den Inferenzmechanismus dar.

[4] $\neg C(x)$ wird wie folgt umgesetzt: $\mathcal{A}'_D := (\mathcal{A}_D \setminus \{\neg C(x)\}) \cup \{C^-(x)\}$, $\mathcal{T}'_D := \mathcal{T}_D \cup \{(C^- \sqsubseteq \neg C), (\neg C \sqsubseteq C^-)\}$, wobei C^- neu in D.

[5] $\neg R(x_1, \cdots, x_n)$ wird wie folgt umgesetzt: $\mathcal{A}'_D := (\mathcal{A}_D \setminus \{\neg R(x_1, \cdots, x_n)\}) \cup \{R^-(x_1, \cdots, x_n)\}$, $\mathcal{T}'_D := \mathcal{T}_D \cup \{(R^- \sqsubseteq \neg R), (\neg R \sqsubseteq R^-)\}$, wobei R^- neu in D.

3.4) und Individuen (Def. 3.7) erzwingen. Dies nutzt man z.B. zur Einschränkung möglicher Modelle (Def. 3.10) des Schemas in Beweisen, etwa um die (Nicht-)Existenz von Objekten mit bestimmten Eigenschaften zu garantieren.

Der Begriff der *Klasse* (*class*) im objektorientierten Ansatz wird in der *Description Logic* durch den Begriff des *Konzepts* (*concept*) ausgedrückt. Ein Konzept ist eine formale Beschreibung der Klasse durch notwendige Bedingungen, denen ein Objekt genügen muss, um Mitglied der Klasse zu sein. Ein Individuum — so bezeichnet man ein Objekt im Kontext von *Description Logics* — das den Anforderungen der Konzeptbeschreibung genügt und Mitglied der entsprechenden Klasse ist, gehört zur *Extension* des zugehörigen Konzepts.

Konzepte werden entweder als primitive (oder atomare) Konzepte deklariert oder sie werden mittels einer *Konzeptbeschreibung* unter Anwendung von *Konstruktoren* erzeugt.

Definition 3.3 (Konzepte). *Konzepte sind Relationen der Stelligkeit 1. Jedem in Schema D deklarierten* primitiven *oder* atomaren Konzept *wird ein Bezeichner $A \in \mathcal{C}_D$ zugeordnet. Weiterhin werden Konzepte durch* Konzeptausdrücke *oder* allgemeine Konzepte *C beschrieben, die folgender Grammatik genügen:*

$$C \quad ::= \quad A \mid \neg C_1 \mid C_1 \sqcap C_2 \mid \top_1 \mid (\leq k\ [i]R) \mid \exists E.C_1$$

Dabei seien C_1, C_2 allgemeine Konzepte, A ein primitives Konzept, R eine Relation, $i \leq arity(R)$ eine natürliche Zahl, k eine nicht-negative, ganze Zahl und E ein regulärer Ausdruck. Die Benennung der Konstruktoren ist in Tabelle 3.1 aufgeführt.

Es gibt zwei ausgezeichnete Konzepte, nämlich

- das Allgemeinste Konzept (`TOP-CONCEPT`) $\top_1$ (abgekürzt $\top$) und
- das Leere Konzept (`NOTHING`) $\bot$ (vgl. Tabelle 3.2).

Relationen (*relations*) formalisieren die Beschreibung von Beziehungen zwischen Individuen. Sie dienen unter anderem zur Erzeugung von Konzeptbeschreibungen mittels Konzeptkonstruktoren. Jeder im Schema D deklarierten primitiven Rolle wird ein Relationsbezeichner $P \in \mathcal{R}_D$ zugeordnet.

Definition 3.4 (Relationen). *Eine* Relation *R wird als Relation mit einer Stelligkeit $n = arity(R) \geq 2$ definiert. Die höchste Stelligkeit einer im Schema vorkommenden Relation sei n_{max} Eine Relation kann entweder als* primitive Relation *mit einem Bezeichner $P \in \mathcal{R}_D$ eingeführt werden oder mit Hilfe eines* Relationsausdrucks *beschrieben werden, welcher folgender Grammatik genügt:*

$$R \quad ::= \quad P \mid \neg R_1 \mid R_1 \sqcap R_2 \mid \top_n \mid (i/n : C)$$

Dabei seien R_1, R_2 Relationsausdrücke, P eine primitive Relation, C ein Konzeptausdruck und i, n natürliche Zahlen mit $1 \leq i \leq n \leq n_{max}$. Zur Benennung der Konstruktoren sei auf Tabelle 3.1 verwiesen.

$\top_n$ bezeichnet für jede Stelligkeit n eine speziellen Relation namens `TOP-RELATION` der Stelligkeit n.

Der Begriff der „Relation“ verallgemeinert hier den ansonsten für *Description Logic*-Sprachen typischen Begriff der *Rolle*. Rollen sind im Allgemeinen ausschließlich binäre Relationen.

Definition 3.5 (Reguläre Ausdrücke). *Reguläre Ausdrücke (*regular expressons*) sind binäre Relationen, die ausschließlich durch entsprechende Konstruktoren (siehe Tabelle 3.6) erzeugt werden können. Insbesondere können reguläre Ausdrücke nicht die linke oder rechte Seite einer Relationsinklusionszusicherung bilden. Ihnen kann kein Bezeichner zugeordnet werden. Reguläre Ausdrücke genügen somit folgender Grammatik:*

$$E \quad ::= \quad \epsilon \mid R|_{[i][j]} \mid E_1 \sqcup E_2 \mid E_1 \circ E_2 \mid E_1^*$$

Dabei seien E_1, E_2 reguläre Ausdrücke, R eine Relation und i, j natürliche Zahlen mit $1 \leq i, j \leq arity(R)$. Zur Benennung der Konstruktoren sei wieder auf Tabelle 3.1 verwiesen.

Reguläre Ausdrücke können ausschließlich in anderen regulären Ausdrücken oder über den Konstruktor „Qualifizierte existentielle Quantifizierung über reguläre Ausdrücke“ ($\exists E.C$) in einem Konzeptkonstruktor auftreten. Da es keine „primitiven regulären Ausdrücke“ gibt und sie niemals Bestandteil einer Relationsinklusionszusicherung sein können, bleiben sie außerdem „anonym“.

Zur Erzeugung komplexer Konzept- und Relationsausdrücke aus bereits vereinbarten Konzept- und Relationsausdrücken dienen *Konstruktoren* (*constructors*). Sie erhalten als Argumente Konzeptausdrücke, Relationsausdrücke und/oder weitere Parameter und bilden ihrerseits einen Konzept- bzw. Relationsausdruck.

Definition 3.6 (Konstruktoren). *Es bezeichnen*

k, i, n	*nicht-negative, ganze Zahlen,*
C, C_i	*Konzeptausdrücke,*
R, R_i	*Relationsausdrücke,*
E, E_i :	*Reguläre Ausdrücke*
$t[i]$	*die i-te Komponente des Tupels t.*

Dann werden Konstruktoren bzw. Konstruktorausdrücke für Konzepte und Relationen sowie abkürzende Notationsformen wie in Tabelle 3.1 aufgeführt vereinbart.

Erläuterungen zu bestimmten Konstruktoren:

Der Ausdruck $(\leq k\ [i]R)$ wird folgendermaßen interpretiert werden: Er bezeichnet ein Konzept, zu dem jedes Individuum o gehört, welches in der i-ten Komponente der interpretierten Relation R höchstens k-mal auftritt (d.h. in der interpretierten

Konzeptkonstruktorausdrücke	
$\top$	(Allgemeinstes Konzept `TOP-CONCEPT`)
$C_1 \sqcap C_2$	(Durchschnitt von Konzepten)
$\neg C$	(„Negation“ für Konzepte)
$(\leq k\ [i]R),\ i \leq \text{arity}(R)$	(Anzahlbeschränkung bezüglich Relation R)
$\exists E.C$	(Qualifizierte existentielle Quantifizierung über reguläre Ausdrücke)
Relationskonstruktorausdrücke	
$\top_n$	(Allgemeinste n-äre Relation `TOP-RELATION`)
$R_1 \sqcap R_2$	(Durchschnitt von Relationen mit $\text{arity}(R_1) = \text{arity}(R_2)$)
$\neg R$	(„Negation“ für Relationen)
$(i/n : C),\ i \leq n$	(Konzeptgebundene Relationskonstruktion)
Konstruktoren für reguläre Ausdrücke	
ϵ	(Identität)
$R\|_{[i],[j]},\ i,j \leq \text{arity}(R)$	(Projektion einer Relation auf zwei ihrer Komponenten)
$E_1 \circ E_2$	(Verkettung regulärer Ausdrücke)
$E_1 \sqcup E_2$	(Vereinigung von regulären Ausdrücken)
E^*	(Reflexiv-Transitiver Abschluss)

Tabelle 3.1: Konzept- und Relationskonstruktoren für $\mathcal{DLR}_{ifd}$ nach [CGL01], erweitert um Reguläre Ausdrücke aus $\mathcal{DLR}_{reg}$ nach [CGL98]

Relation R sind höchstens k Tupel enthalten, in deren i-ter Komponente sich das Individuum o findet).

Der Ausdruck $(i/n : C)$ wird wie folgt interpretiert werden: Er bezeichnet eine n-stellige Relation, die unter einer Interpretation jedes Tupel enthält, dessen i-te Komponente unter derselben Interpretation zur Extension des Konzepts C gehört.

3.2.2 Semantik von $\mathcal{DLR}_{reg,ifd}$

Um eine mittels eines Schemas formal beschriebene Wissensbasis mit „Bedeutung“ zu füllen, bedient man sich einer Interpretation. Diese füllt zunächst das „Universum der möglichen Objekte“ (die Interpretationsdomäne) mit Objekten und legt dann die Extensionen aller Konzepte und die den einzelnen Relationen zugehörigen Tupel mittels einer Interpretationsfunktion fest.

Definition 3.7 (Interpretationsdomäne und Individuen). *Die* Interpretationsdomäne *(interpretation domain)* $\Delta^{\mathcal{I}} \subset \mathcal{F} \setminus (\mathcal{C}_D \cup \mathcal{R}_D)$ *ist eine Menge von „Objekten“. Die Elemente von* $\Delta^{\mathcal{I}}$ *heißen* Individuen *(individuals). Individuen werden mit ihrem identifizierenden Bezeichner beschrieben.*

Abkürzende Notationsformen	
$\bot = \neg\top$	(Leeres Konzept NOTHING)
$C_1 \sqcup C_2 = \neg(\neg C_1 \sqcap \neg C_2)$	(Vereinigung von Konzepten)
$C_1 \Rightarrow C_2 = \neg C_1 \sqcup C_2$	(Implikation von Konzepten)
$C_1 \setminus C_2 = C_1 \sqcap (\neg C_2)$	(Differenz für Konzepte)
$R_1 \sqcup R_2 = \neg(\neg R_1 \sqcap \neg R_2)$	(Vereinigung von Relationen)
falls $\text{arity}(R_1) = \text{arity}(R_2) = n, R_1^{\mathcal{I}} \cup R_2^{\mathcal{I}} \subseteq \top_n^{\mathcal{I}}$	
$(\geq k\,[i]R) = \neg(\leq k-1\,[i]R)$	(alternative Anzahlbeschränkung)
$\exists\,[i]R = (\geq 1\,[i]R)$	(unqualifizierte existentielle Quantifizierung)
$\forall\,[i]R = \neg\exists\,[i](\neg R)$	(unqualifizierte universale Quantifizierung)
$\forall\,E.C = \neg(\exists E.\neg C)$	(Werterestriktion für reguläre Ausdrücke)
$\text{id}(C) = (1/2 : C)\vert_{[1][1]}$	(Identität auf Konzept C)
$R^{-} = R\vert_{[2][1]}$, arity(R)=2	(„Inverse Rolle“)

Tabelle 3.2: Notationsweisen für $\mathcal{DLR}_{ifd}$ und $\mathcal{DLR}_{reg}$ nach [CGL01, CGL98]

Individuen bezeichnen Seiende bzw. Objekte innerhalb einer Instanz eines Schemas. Sie können einem Konzept oder mehreren Konzepten entsprechen und gehören dann zur Extension des betreffenden Konzepts. Individuen haben eine von ihren Eigenschaften unabhängige Existenz. Ebenfalls dürfen unterschiedliche Individuen bezüglich sämtlicher Eigenschaften übereinstimmen (es sei denn, dies wird durch entsprechende semantische Bedingungen (z.B. Schlüsselbedingungen) ausgeschlossen).

Definition 3.8 (Interpretationsfunktion). *Es bezeichnen*

k, i, n	*nicht-negative, ganze Zahlen,*
C, C_i	*Konzeptausdrücke,*
R, R_i	*Relationsausdrücke,*
$t[i]$	*die i-te Komponente des Tupels t.*

Eine Interpretationsfunktion *(interpretation function) zu einem Schema D und einer Interpretationsdomäne $\Delta^{\mathcal{I}}$ ist eine Funktion $\cdot^{\mathcal{I}}$, die syntaktischen Strukturen Ausdrücke über der Interpretationsdomäne $\Delta^{\mathcal{I}}$ zuordnet. Insbesondere ordnet $\cdot^{\mathcal{I}}$*

- *jedem primitiven Konzept $C \in \mathcal{C}_D$ seine Extension $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,*
- *jeder TOP-ROLE-n $\top_n$ eine Teilmenge $\top_n^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$,*
- *jeder primitiven Relation $R \in \mathcal{R}_D$ eine Teilmenge von $\top_n^{\mathcal{I}}$, für arity$(R) = n$,*
- *jeder der in $ABox_D$ auftretenden Skolemkonstanten x ein Individuum $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,*
- *jedem nicht-primitiven, d.h. Konstruktorausdruck, einen Ausdruck nach Tabelle 3.3,*
- *jedem regulären Ausdruck E eine Teilmenge $E^{\mathcal{I}}$ von $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ zu.*

Die beiden vorangegangenen Definitionen bilden die Grundlage für die nun folgende Definition der Interpretation zu einem Schema.

Definition 3.9 (Interpretation). *Eine* Interpretation *zu einem Schema K ist ein Tupel $\mathcal{I} = \langle \Delta^{\mathcal{I}}, .^{\mathcal{I}} \rangle$, wobei $\Delta^{\mathcal{I}}$ eine Interpretationsdomäne und $.^{\mathcal{I}}$ eine Interpretationsfunktion zu K und $\Delta^{\mathcal{I}}$ ist.*

Interpretationen können Bedingungen, die durch das Schema der Wissensbank in der Terminologie der TBox und den Zusicherungen der ABox festgelegt sind, erfüllen oder nicht erfüllen. Schließlich bezeichnet man eine Interpretation, die sämtliche im Schema enthaltenen Bedingungen erfüllt, als Modell oder Instanz der Wissensbasis:

Definition 3.10 (Modell). *Die Interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, .^{\mathcal{I}} \rangle$ eines Schemas $K = \langle E_K | \mathcal{T}_K | \mathcal{A}_K | \mathcal{S}_K \rangle$* erfüllt eine Zusicherung α, *wenn gilt:*

- $\alpha \in \mathcal{T}_K$ ist eine Inklusionszusicherung:
  Wenn $\alpha = (C_1 \sqsubseteq C_2)$, *so gilt* $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
  Wenn $\alpha = (R_1 \sqsubseteq R_2)$, *so gilt* $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
- $\alpha \in \mathcal{A}_K$ ist eine ABox-Zusicherung:
  Für Skolemkonstanten x, x_i $(1 \leq i \leq n)$, y *gilt:*
  Wenn $\alpha = (C(x))$, *so gilt* $x^{\mathcal{I}} \in C^{\mathcal{I}}$;

Ausdruck	Interpretation
$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
$\top_n$	$\top_n^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$
$A, A \in \mathcal{C}_D$	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$P, P \in \mathcal{R}_D$	$P^{\mathcal{I}} \subseteq \top_n^{\mathcal{I}}, n = \text{arity}(P)$
$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
$R_1 \sqcap R_2, \text{arity}(R_1) = \text{arity}(R_2)$	$(R_1 \sqcap R_2)^{\mathcal{I}} = R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$
$\neg R, n = \text{arity}(R)$	$(\neg R)^{\mathcal{I}} = \top_n^{\mathcal{I}} \setminus R^{\mathcal{I}}$
$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$(\leq k\ [i]R),\ i \leq \text{arity}(R)$	$(\leq k\ [i]R)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \sharp\{t \in R^{\mathcal{I}} \mid t[i] = d\} \leq k\}$
$(i/n : C),\ i \leq n$	$(i/n : C)^{\mathcal{I}} = \{t \in \top_n^{\mathcal{I}} \mid t[i] \in C^{\mathcal{I}}\}$
x, Skolemkonstante	$x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
ϵ	$\epsilon^{\mathcal{I}} = \{(x, x) \mid x \in \Delta^{\mathcal{I}}\}$
$R\|_{[i],[j]},\ \text{arity}(R) = n \geq i, j$	$(R\|_{[i],[j]})^{\mathcal{I}} = \{(x_i, x_j) \mid (x_1, \cdots, x_n) \in R^{\mathcal{I}}\}$
$E_1 \circ E_2$,	$(E_1 \circ E_2)^{\mathcal{I}} = E_1^{\mathcal{I}} \circ E_2^{\mathcal{I}}$
$E_1 \sqcup E_2$	$(E_1 \sqcup E_2)^{\mathcal{I}} = E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}}$
E^*	$(E^*)^{\mathcal{I}} = (E^{\mathcal{I}})^*$
$\exists E.C$	$(\exists E.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists d' \in C^{\mathcal{I}} : (d, d') \in E^{\mathcal{I}}\}$

Tabelle 3.3: Interpretationsfunktion $.^{\mathcal{I}}$ zu Tabelle 3.1

Anmerkung zu Tabelle 3.3: $\top_n^{\mathcal{I}}$ bezeichnet nicht das n-fache Kartesische Produkt von $(\Delta^{\mathcal{I}})$, sondern eine Teilmenge desselben, die alle n-stelligen Relationen des Schemas überdeckt. Somit bezeichnet $(\neg R)^{\mathcal{I}}$ auch nicht ein Komplement, sondern eine Differenz.

wenn $\alpha = (R(x_1, \ldots, x_n))$, *so gilt* $(x_1^{\mathcal{I}}, \ldots, x_n^{\mathcal{I}}) \in R^{\mathcal{I}}$;
wenn $\alpha = (x = y)$, *so gilt* $x^{\mathcal{I}} = y^{\mathcal{I}}$;
wenn $\alpha = (x \neq y)$, *so gilt* $x^{\mathcal{I}} \neq y^{\mathcal{I}}$.

- $\alpha \in \mathcal{S}_K$ ist Zusicherung einer semantischen Bedingung:

  Wenn $\alpha = (\textbf{\textit{id}}\ C\ [i_1]R_1, \ldots, [i_h]R_h)$
  (α ist eine Identitätszusicherung (ID-Zusicherung, identification assertion*)),*
  so gilt für alle Individuen $a, b \in C^{\mathcal{I}}$ *und alle Tupel* $t_1, s_1 \in R_1^{\mathcal{I}}, \ldots, t_h, s_h \in R_h^{\mathcal{I}}$*:*

  $$\left.\begin{array}{l} a = t_1[i_1] = \cdots = t_h[i_h], \\ b = s_1[i_1] = \cdots = s_h[i_h], \\ t_j[i] = s_j[i],\ \textit{für}\ j \in \{1, \ldots, h\}\ \textit{und}\ i \neq i_j \end{array}\right\} \textit{impliziert}\ a = b.$$

  Wenn $\alpha = (\textbf{\textit{fd}}\ R\, i_1, \ldots, i_h \rightarrow i)$
  (α ist eine Zusicherung einer funktionalen Abhängigkeit (FD-Zusicherung, functional dependency assertion*)),*
  so gilt für alle Tupel $t, s \in R^{\mathcal{I}}, h > 1$*:*

  $$t[i_1] = s[i_1], \ldots, t[i_h] = s[i_h]\ \textit{impliziert}\ t[i] = s[i].$$

Eine Interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, .^{\mathcal{I}} \rangle$ *heißt* Modell *(model) oder* Instanz *(instance) eines Schemas* $K = \langle E_K | \mathcal{T}_K | \mathcal{A}_K | \mathcal{S}_K \rangle$*, wenn sie alle im Schema formulierten Zusicherungen* $\alpha \in \mathcal{T}_K \cup \mathcal{A}_K \cup \mathcal{S}_K$ *erfüllt.*

Ein Schema K *heißt* erfüllbar *(satisfiable) gdw. es eine Interpretation* $\mathcal{I}$ *gibt, die Modell von* K *ist.*

Ein Schema K *heißt* endlich erfüllbar*, wenn es eine Interpretation mit endlicher Interpretationsdomäne gibt, die das Schema erfüllt. Eine solche Interpretation heißt dann* endliches Modell *(finite model) von* K*.*

Auf Grundlage eines gegebenen Schemas mit seinen expliziten Bedingungen und des Begriffs der Erfüllbarkeit des Schemas kann man versuchen, Schlüsse auf weitere, implizite Bedingungen zu ziehen. Dies wird mit dem Begriff der logischen Implikation formalisiert.

Definition 3.11 (Logische Implikation). *Sei* K *ein Schema. Eine Inklusionszusicherung oder Zusicherung einer semantische Bedingung, etwa* $\alpha = (C_1 \sqsubseteq C_2)$*,* folgt logisch aus K, notiert $(K \models \alpha)$*, gdw. jedes Modell von* K *auch* α *erfüllt.*

Ist Σ *eine TBox, so folgt* α *logisch aus* $K \cup \Sigma$*, notiert* $(K \cup \Sigma \models \alpha)$*, genau dann wenn jedes Modell von* $K \cup \Sigma$ *auch* α *erfüllt.*

Es sind vor allem zwei Probleme des logischen Folgerns, die regelmäßig zu lösen sind: Das *Konzeptkonsistenz- oder Erfüllbarkeitsproblem* („Gilt in einem Schema K für ein Konzept C die logische Folgerung $K \not\models (C \sqsubseteq \bot)$?“) und das *Konzeptenthaltenseins- oder Folgerungsproblem* („Gilt in einem Schema K für Konzepte C_1, C_2 $K \models (C_1 \sqsubseteq C_2)$?“).

Satz 3.1 (Äquivalenz von logischer Implikation und Erfüllbarkeit in $\mathcal{DLR}_{reg,ifd}$). *Das Problem der logischen Implikation und das Erfüllbarkeitsproblem sind in der hier betrachteten Description Logic aufeinander reduzierbar:*

(1) K *ist unerfüllbar gdw.* $K \models (\top_1 \sqsubseteq \bot)$.

(2) $K \models C_1 \sqsubseteq C_2$ *gdw.* $K \cup \{\top_1 \sqsubseteq \exists[1](P \sqcap (2/2 : C_1 \sqcap \neg C_2))\}$ *ist unerfüllbar, wobei* P *eine neu in* K *eingeführte binäre Relation ist.*

(3) $K \models R_1 \sqsubseteq R_2$ *gdw.* $K \cup \{\top_1 \sqsubseteq \exists[1](P \sqcap (2/2 : \exists[1](C_1 \sqcap \neg C_2)))\}$ *ist unerfüllbar, wobei* P *eine neu in* K *eingeführte binäre Relation ist.*

(4) $K \models C(x)$ *gdw.* $K \cup \{\neg C(x)\}$ *ist unerfüllbar.*

(5) Zusicherungen für semantische Bedingungen (FD- und ID-Zusicherungen) können unter Konstruktion einer geeigneter Erweiterung von $ABox_K$ *ähnlich behandelt werden (vgl. Abschnitt 3.3.1) .*

Beweis: *Der Beweis beruht auf der Abgeschlossenheit der Konzepte unter den Bool'schen Operationen. Zu Details siehe [GL96, CGL98, CGL01].*

3.3 Entscheidungsverfahren für $\mathcal{DLR}_{reg,ifd}$

Das Verfahren zur Entscheidung der logischen Implikation in der in Abschnitt 3.2 vorgestellten *Description Logic* $\mathcal{DLR}_{reg,ifd}$ ist in der Literatur nirgendwo vollständig beschrieben. Stattdessen gibt es eine Reihe von teils aufeinander aufbauenden einzelnen Veröffentlichungen [CGL95, GL96, CGL98, CGL01]. Daher werden die Teilergebnisse der Forschung hierzu und das Verfahren an dieser Stelle in einer zusammenfassenden und einheitlichen Form vorgestellt. Das Entscheidungsverfahren besteht aus vier Schritten, die nachfolgend im Einzelnen dargestellt werden:

(1) Reduktion der logischen Implikation in $\mathcal{DLR}_{reg,ifd}$ auf die Wissensbasis-Erfüllbarkeit in $\mathcal{DLR}_{reg}$ (Unterabschnitt 3.3.1)

(2) Reduktion der Erfüllbarkeit einer $\mathcal{DLR}_{reg}$-Wissensbasis auf die $\mathcal{DLR}_{reg}$-Konzepterfüllbarkeit (Unterabschnitt 3.3.2)

(3) Reduktion des Erfüllbarkeitsproblems in $\mathcal{DLR}_{reg}$ auf das Erfüllbarkeitsproblem in PDL (Unterabschnitt 3.3.3)

(4) Entscheidungsverfahren für das Erfüllbarkeitsproblem in PDL (Unterabschnitt 3.3.4).

3.3.1 Reduktion der logischen Implikation in $\mathcal{DLR}_{reg,ifd}$ auf Wissensbasis-Erfüllbarkeit in $\mathcal{DLR}_{reg}$

Calvanese et al. zeigen, dass die logische Implikation in $\mathcal{DLR}_{reg,ifd}$-Wissensbasen auf die Wissensbasis-Erfüllbarkeit von $\mathcal{DLR}_{reg}$-Wissensbasen reduziert werden kann [CGL01]. Dies geschieht folgendermaßen: Jedes Implikationsproblem in $\mathcal{DLR}_{ifd}$ hat eine der folgenden Formen:

(1) $K \models C_1 \sqsubseteq C_2$ (Implikation einer CIC),

(2) $K \models R_1 \sqsubseteq R_2$ (Implikation einer RIC),

(3) $K \models C(\alpha)$ (Implikation einer Instanz-Zusicherung),

(4) $K \models (\mathbf{id}\ C :\ [i_1]R_1, \ldots, [i_h]R_h)$ (Implikation einer ID-Zusicherung),

(5) $K \models (\mathbf{fd}\ R :\ i_1, \ldots, i_h \rightarrow i)$ (Implikation einer FD-Zusicherung).

Diese Implikationsprobleme lassen sich jeweils wie folgt auf Wissensbasis-Erfüllbarkeitsprobleme für $\mathcal{DLR}_{reg}$ zurückführen (wobei P eine in K neue, primitive, binäre Relation sei):

(1) CICs: $K \models C_1 \sqsubseteq C_2$ gdw.
$K \cup \{\top_1 \sqsubseteq \exists[1](P \sqcap (2 : C_1 \sqcap \neg C_2))\}$ ist unerfüllbar [GL96, CGL98],

(2) RICs: $K \models R_1 \sqsubseteq R_2$ gdw.
$K \cup \{\top_1 \sqsubseteq \exists[1](P \sqcap (2 : \exists[1](R_1 \sqcap \neg R_2)))\}$ ist unerfüllbar [GL96, CGL98][6],

(3) $K \models C(\alpha)$ gdw.
$K \cup \{\neg C(\alpha)\}$ ist unerfüllbar [GL96, CGL98],

(4) ID-/FD-Zusicherungen: Erweiterung von ABox_K um konkrete Gegenbeispiele. Test auf Unerfüllbarkeit des Gegenbeispiels. [CGL01].

Für Wissensbasen ohne ID- und FD-Zusicherungen (das sind $\mathcal{DLR}_{reg}$-Wissensbasen) folgt dies unmittelbar aus den Ergebnissen für diese Probleme in der *Description Logic*-Sprache $\mathcal{CIQ}$, die $\mathcal{DLR}_{reg}$ ohne n-äre Relationen (also mit ausschließlich binären Rollen) entspricht [GL96, CGL98]. Die Erweiterung von $\mathcal{CIQ}$ auf $\mathcal{DLR}_{reg}$ besteht im Wesentlichen in der Einführung n-ärer Relationen und RICs durch ein repräsentierendes atomares Konzept für jede Relation und atomarer, funktionaler Rollen für die Darstellung der Relationstupel (vgl. „Reifikation“, Seite 51).

[6] In [CGL01] lautet es fälschlich: $K \models R_1 \sqsubseteq R_2$ gdw. $K \cup \{\top_1 \sqsubseteq \exists[1](P \sqcap (2 : \exists[1](C_1 \sqcap \neg C_2)))\}$ ist unerfüllbar.

Für die ID-Zusicherungen und FD-Zusicherunegn zeigen Calvanese et al. diese Äquivalenzen in (4) wie folgt [CGL01]: Zu einer binären Relation R sind *identification assertions* der Form (**id** C $[i]R$) äquivalent zu $\mathcal{DLR}$-Zusicherungen $\top \sqsubseteq (\leq 1[j](R \sqcap (I : C)))$ (wobei $j = 2$, falls $i = 1$, und $j = 1$, falls $i = 2$). Derartige ID-Zusicherungen werden daher ferner nicht mehr berücksichtigt.

Zu einer ID-Zusicherung $\kappa = (\textbf{id}\ C\ [i_1]R_1, \ldots, [i_h]R_h)$ wird nun eine ABox A_κ folgendermaßen konstruiert:

- $C(x), C(y), x \neq y$, mit neuen Skolemkonstanten x, y;
- $R_j(t_j), R_j(s_j)$, mit $j \in \{1, \ldots, h\}$, t_j und s_j sind Tupel von neuen Skolemkonstanten mit $t_j[i_j] = x, s_j[i_j] = y$ und $t_j[i] = s_j[i]$ für $i \neq i_j$.

Zu einer FD-Zusicherung $\kappa = (\textbf{fd}\ R\ i_1, \ldots, i_h \rightarrow j)$ wird eine ABox A_κ folgendermaßen konstruiert:

- $R(t), R(s), t[j] \neq s[j]$, wobei s, t Tupel von neuen Skolemkonstanten mit $t[i_j] = s[i_j]$ für $j \in \{1, \ldots, h\}$ sind.

A_κ liefert jeweils ein Gegenbeispiel für κ. Somit gilt $K \models \kappa$ gdw. $K \cup A_\kappa$ unerfüllbar ist.

Als eine Entscheidungsprozedur zur Überprüfung der Erfüllbarkeit einer $\mathcal{DLR}_{ifd}$ - Wissensbasis K geben Calvanese et al. folgendes Verfahren an [CGL01]:

Eine Wissensbasis $K = \langle E_K | \mathcal{T}_K | \emptyset | \mathcal{S}_K \rangle$ mit leerer ABox ist genau dann erfüllbar, wenn $TBox_K$ erfüllbar ist. Die in diesem Fall geltende *tree model property* garantiert, dass ID-/FD-Zusicherungen in trivialer Weise erfüllt werden, da zwei Tupel in einem solchen Modell maximal in einer Komponente übereinstimmen können.

Im Falle einer nicht-leeren ABox geht die *tree model property* verloren. Allerdings können Modelle gefunden werden, die weitestgehend eine Baumstruktur aufweisen. Ausnahmen innerhalb dieser Modelle werden von Clustern von Objekten gebildet, die Skolemkonstanten aus der ABox repräsentieren. Derartige Modelle heißen *clustered tree models*. Darin sind ID/FD-Zusicherungen in trivialer Weise erfüllt, mit Ausnahme der möglichen Cluster von Skolemkonstanten. Daher kann man sich in *clustered tree models* bei der Überprüfung von ID/FD-Zusicherungen auf Tupel beschränken, die in der ABox auftreten.

Dazu konstruieren Calvanese et al. eine *Saturierung* von K als eine ABox $\mathcal{A}_s$ folgendermaßen:

- Für jede Skolemkonstante x, die in K auftritt, und für jede ID-Zusicherung (**id** $C\ [i_1]R_1, \ldots, [i_h]R_h$) in K enthält $\mathcal{A}_s$ entweder $C(x)$ oder $\neg C(x)$.

- Für jedes Tupel t von Skolemkonstanten, das in einer Zusicherung $R(t)$ in K auftritt:
  - Für jede ID-Zusicherung (**id** C $[i_1]R_1, \ldots, [i_h]R_h$) in K: Für jedes R_j, ($j \in \{1, \ldots, h\}$) mit der Stelligkeit von R enthält $\mathcal{A}_s$ entweder $R_j(t)$ oder $\neg R_j(t)$;
  - für jede FD-Zusicherung (**fd** R' $i_1, \ldots, i_h \rightarrow j$) in K, so dass R' die Stelligkeit von R hat: $\mathcal{A}_s$ enthält entweder $R'(t)$ oder $\neg R'(t)$.
- Für jedes Paar von Skolemkonstanten x und y, die in K auftreten, enthält $\mathcal{A}_s$ entweder $x = y$ oder $x \neq y$.

Die Größe jeder der exponentiell vielen möglichen Saturierungen $\mathcal{A}_s$ ist polynomiell in der Größe von K. An einer Saturierung kann man mit polynomiellem Aufwand unmittelbar überprüfen, ob eine ID- oder FD-Zusicherung verletzt wird. Nach der Überprüfung der ID-/FD-Zusicherungen ist noch die Konsistenz von $\mathcal{A}_s$ mit den restlichen Zusicherungen aus K zu prüfen:

Eine Wissensbasis $K = \langle E_K | \mathcal{T}_K | \mathcal{A}_K | \mathcal{S}_K \rangle$ ist genau dann erfüllbar, wenn eine Saturierung A_s von K existiert, die keine der ID- und FD-Zusicherungen in $\mathcal{S}_K$ verletzt, und die $\mathcal{DLR}$- Wissensbasis $\langle E_K | \mathcal{T}'_K | \mathcal{A}_K \cup \mathcal{A}_s \rangle$ erfüllbar ist (Beweis: [CGL01]).

Mittels Testen höchstens exponentiell vieler Saturierungen $\mathcal{A}_s$ auf Erfüllen aller ID- und FD-Zusicherungen und dem Überprüfen der Erfüllbarkeit der $\mathcal{DLR}_{reg}$-Wissensbasis $\langle E_K | \mathcal{T}'_K | \mathcal{A}_K \cup A_s \rangle$ (ein exponentieller Schritt, vgl. 3.3.4) ist somit die Erfüllbarkeit einer $\mathcal{DLR}_{reg,ifd}$-Wissensbasis K mit insgesamt exponentiellem Rechenaufwand entscheidbar.

Zwar zeigen Calvanese et al. die soeben vorgestellte Reduktion in [CGL01] nur für die Rückführung der DL $\mathcal{DLR}_{ifd}$ (das ist $\mathcal{DLR}_{ifd,reg}$ ohne reguläre Ausdrücke und die damit einhergehenden Konstruktoren) auf $\mathcal{DLR}$, allerdings merken sie an, dass man das Ergebnis unmittelbar auf eine Erweiterung von $\mathcal{DLR}_{reg}$ um ID- und FD-Zusicherungen (das ist gerade $\mathcal{DLR}_{ifd,reg}$) übertragen kann, wobei die logische Implikation DEXPTIME-vollständig bleibt.

3.3.2 Reduktion der Erfüllbarkeit einer $\mathcal{DLR}_{reg}$-Wissensbasis auf die $\mathcal{DLR}_{reg}$-Konzepterfüllbarkeit

Die Behandlung von ABox-Zusicherungen stellt für die Lösung von Erfüllbarkeitsproblemen in DL kein geringes Problem dar. Für Wissensbasen mit azyklischer TBox kann das Problem durch iteriertes Ersetzen sämtlicher Konzeptbezeichner durch ihre Definition auf ein Erfüllbarkeitsproblem einer Wissensbasis mit leerer TBox reduziert werden [DLNS94].

Während De Giacomo zunächst für die *Description Logic*-Sprachen $\mathcal{DIO}$ und $\mathcal{DNO}$ Verfahren zur Entscheidung von Erfüllbarkeit und logischer Implikation in

Sprachen mit freier (d.h. potenziell zyklischer) TBox und entweder inversen Rollen *oder* Anzahlbeschränkungen vorstellt [Gia95], betrachten De Giacomo und Lenzerini die Behandlung von Zusicherungen für Individuen (ABox-Zusicherungen) in DL mit freier TBox, inversen Rollen *und* Anzahlbeschränkungen [GL96]. Hier wird eine DL namens $\mathcal{CIQ}$ eingeführt, deren Schemata TBoxes und ABoxes aufweisen. Im Gegensatz zu den $\mathcal{DLR}$-Dialekten enthalten ABoxes in $\mathcal{CIQ}$ jedoch keine Skolemkonstanten, sondern Individuenkonstanten, d.h. $.^{\mathcal{I}}$ interpretiert Individuenkonstanten immer so, dass unterschiedliche Konstanten unterschiedliche Individuen vertreten (also: $C(x), C(y) \Rightarrow x \neq y$).[7] Zyklen in der TBox werden ausdrücklich zugelassen. Das Problem der $\mathcal{CIQ}$-Wissensbasiserfüllbarkeit ist in EXPTIME. $\mathcal{CIQ}$ kennt keine Negation von Rollen und keinen rollenbasierten Konzeptkonstruktor. Anzahlbeschränkungen sind nur für atomare Rollen erlaubt. Insofern ist die vorgestellte Behandlung von ABox-Zusicherungen in [GL96] exemplarisch zu betrachten und auf [CGL95] zu übertragen.

Die Erfüllbarkeit einer $\mathcal{CIQ}$-Wissensbasis wird zunächst polynomiell auf die Erfüllbarkeit einer $\mathcal{CIQ}$-Wissensbasis mit einer einzigen Instanzzusicherung $C(x)$ in der ABox reduziert (*Internalisierung einer ABox*, vgl. auch [SCM02]). Dieses Problem ist identisch mit der Konzept-Erfüllbarkeit dieses einen Konzeptes C in Gegenwart einer $\mathcal{CIQ}$-TBox. Dieses Problem schließlich ist entscheidbar mittels eines bekannten Algorithmus' zur Entscheidung des Unerfüllbarkeitsproblems für Konzepte und EXPTIME-vollständig [GL95, Gia95].

3.3.2.1 Verfahren der ABox-Internalisierung

Der *Fischer-Ladner-Abschluss* $CL(C)$ eines Konzepts C entspricht der Menge aller Unterkonzepte von C, das sind sämtliche Konzepte, die bei der Auswertung der Interpretation von C eine Rolle spielen. $CL(C)$ ist die kleinste Menge S von Konzepten, so dass gilt:

$$\begin{aligned}
C_1 \sqcap C_2 \in S &\Rightarrow C_1, C_2 \in S \\
\neg C' \in S &\Rightarrow C' \in S \\
C' \in S &\Rightarrow \neg C' \in S \text{ (falls } C' \neq \neg C'') \\
(\geq nQ.C') \in S &\Rightarrow C' \in S \\
\exists R.C' \in S &\Rightarrow c' in S \\
\exists R_1 \circ R_2.C' \in S &\Rightarrow \exists R_1.\exists R_2.C' \in S \\
\exists R_1 \sqcup R_2.C' in S &\Rightarrow \exists R_1.C, \exists R_2.C' \in S \\
\exists R^*.C' \in S &\Rightarrow \exists R.\exists R^*.C' \in S \\
\exists id(C'').C' \in S &\Rightarrow C'' \in S
\end{aligned}$$

[7] Die Behandlung von $\mathcal{DLR}_{reg}$-ABox-Zusicherungen $(x = y)$ und $(x \neq y)$ ist natürlich problemlos mittels Individuenkonstanten möglich, indem man Skolemkonstanten in Gleichheitszusicherungen miteinander identifiziert und durch nur eine Individuenkonstante ersetzt bzw. die Ungleichheitszusicherung einfach entfällt.

Der Fischer-Ladner-Abschluss einer Wissenbasis oder einer TBox ist die Vereinigung aller $CL(C_i)$ für alle in der Wissenbasis/der TBox vorkommenden Konzepte C_i.

Es sei $\mathfrak{z}$ die *leere Folge von Rollen* mit $\exists\ \mathfrak{z}\ .C := C, \forall\ \mathfrak{z}\ .C := C$. Zu einer Rolle R wird die Menge der Präfixrollen $Pre(R)$ induktiv wie folgt gebildet:

$$\begin{aligned} Pre(Q) &= \{\mathfrak{z}, Q\} \\ Pre(R_1 \circ R_2) &= \{R_1 \circ R_2' | R_2' \in Pre(R_2)\} \cup Pre(R_1) \\ Pre(R_1 \sqcup R_2) &= Pre(R_1) \cup Pre(R_2) \\ Pre(R_1)^* &= \{R_1^* \circ R_1' | R_1' \in Pre(R_1)\} \\ Pre(id(C)) &= \{\mathfrak{z}, id(C)\}. \end{aligned}$$

Zu einer $\mathcal{CIQ}$-Wissensbasis $\mathcal{K} = (TBox, ABox)$ wird die *reduzierte Form* von K, $\mathcal{K}' = (TBox', ABox')$ wie folgt definiert (wobei für jedes Individuum α_i $(i = 1, \ldots, m)$, das in $ABox$ auftritt, ein neues atomares Konzept A_i eingeführt wird).

- $ABox' = \{(\exists create.A_1 \sqcap \ldots \sqcap \exists create.A_m)(g)\}$, wobei g ein neues Individuum (das einzige in $ABox'$) und $create$ eine neue atomare Rolle ist.[8]
- $TBox' = TBox'_{\mathcal{K}} \cup TBox'_{aux}$
  - $TBox'_{\mathcal{K}} = TBox'_{TBox} \cup TBox'_{ABox}$,[9] wobei $TBox'_{TBox} = TBox$ und T'_{Abox} besteht aus den folgenden CICs:

    $$A_i \sqsubseteq C$$

    für jede Instanzzusicherung $C(\alpha_i) \in ABox$, zwei CICs

    $$\begin{aligned} A_i &\sqsubseteq \exists P.A_j \sqcap (\leq 1 P.A_j) \\ A_j &\sqsubseteq \exists P^-.A_i \sqcap (\leq 1 P^-.A - i)) \end{aligned}$$

    für jede Instanzzusicherung $P(\alpha_1, \alpha_2) \in ABox$ sowie eine CIC

    $$A_i \sqsubseteq \sqcap_{i \neq j} \neg A_j$$

    für jedes Individuum α_i, das in $ABox$ auftritt.
  - $TBox'_{aux}$[10] besteht aus einer CIC ($\mathbf{u} := (P_1 \sqcup \ldots \sqcup P_n \sqcup P_1^- \sqcup \ldots \sqcup P_n^-)^*$, wobei die $P_1, \ldots, P_n$ alle in $TBox'_{\mathcal{K}}$ auftretenden atomaren Rollen sind):

    $$(A_i \sqcap C) \sqsubseteq \forall \mathbf{u}.(\neg A_i \sqcup C)$$

    für jedes A_i, das in $TBox'_{\mathcal{K}}$ auftritt und jedes C, so dass gilt:

[8] Dies gestattet es, sich auf Modelle von $\mathcal{K}'$ zu beschränken, die mit g zusammenhängende Graphen repräsentieren.

[9] $TBox'_{\mathcal{K}}$ besteht aus den ursprünglichen TBox-Zusicherungen zuzüglich einer „naiven“ Kodierung der ABox.

[10] $TBox'_{aux}$ gewährleistet, dass für jedes Individuum α_i später genau eine Instanz des Stellvertreterkonzepts A_i als Repräsentant für α_i identifiziert werden kann.

(1) $C \in CL(TBox'_{\mathcal{K}})$
(2) $C = \exists\overline{R}.C'$ mit $\exists R.C' \in CL(TBox'_{\mathcal{K}})$
(3) $C = \exists(\overline{R'} \circ Q).A_j$ mit $R' \in Pre(R), Q = P \mid P^-$ und R, P, A_j treten in $CL(TBox'_{\mathcal{K}})$ auf.

Dabei sei $\overline{R}$ induktiv definiert durch $(Q = P \mid P^-)$:

* $\overline{Q} = Q \circ id(\sqcap_i \neg A_i)$;
* $\overline{R_1 \circ R_2} = \overline{R_1} \circ \overline{R_2}$;
* $\overline{R_1 \sqcup R_2} = \overline{R_1} \sqcup \overline{R_2}$;
* $\overline{R_1^*} = \overline{R_1}^*$;
* $\overline{id(C)} = id(C)$.

Ein Modell $\mathcal{I}$ von $\mathcal{K}$ kann zu einem Modell von $\mathcal{K}'$ erweitert werden, indem das Individuum g zu $\Delta^{\mathcal{I}}$ hinzugefügt wird und die neue Relation *create* wie folgt interpretiert wird: $create^{\mathcal{I}} = \{(g, \alpha_i) | i = 1, \ldots, m\}$. Aus einem Modell $\mathcal{I}$ von $\mathcal{K}'$ kann man andererseits eine Interpretation $\mathcal{I}'$ erzeugen, die alle CICs in $TBox$ erfüllt, in der jedem Individuum α_i in $\mathcal{K}$ genau ein Individuum s_{α_i} in $\mathcal{I}'$ entspricht, so dass $s_{\alpha_i} \in C^{\mathcal{I}}$ für jede Instanzzusicherung $C(\alpha_i)$ in $\mathcal{K}$ und $P(s_{\alpha_i}, s_{\alpha_j}) \in P^{\mathcal{I}'}$ für jede Instanzzusicherung $P(\alpha_i, \alpha_j)$.

Es gilt somit: $\mathcal{K}$ ist genau dann erfüllbar, wenn $\mathcal{K}'$ erfüllbar ist. Somit ist ist das Erfüllbarkeitsproblem für $\mathcal{CIQ}$-Wissensbasen EXPTIME-vollständig.

Die Erfüllbarkeit einer $\mathcal{DLR}_{reg}$-Wissensbasis kann also polynomiell auf die Erfüllbarkeit einer $\mathcal{DLR}_{reg}$-Wissenbasis mit einer einzigen Instanzzusicherung in der ABox reduziert werden. Damit ist das Problem der $\mathcal{DLR}_{reg}$-Wissensbasiserfüllbarkeit identisch mit dem Problem der Konzepterfüllbarkeit ohne Berücksichtigung einer ABox.

3.3.3 Reduktion von Erfüllbarkeitsproblemen in $\mathcal{DLR}_{reg}$ auf Erfüllbarkeitsprobleme in PDL

Der nächste Schritt des Entscheidungsverfahrens besteht in einer Reduktion des TBox-Erfüllbarkeitsproblems für $\mathcal{DLR}_{reg}$ auf ein Erfüllbarkeitsproblem für eine Formel in einer Logik, für die eine Entscheidungsprozedur für das Erfüllbarkeitsproblem bekannt ist. Diese Logik gehört zur Familie der *Propositional Dynamic Logics* (PDL). Genauer handelt es sich um die Variante *Converse PDL with graded modalities* ($CPDL_g$).

Zunächst folgt eine Einführung in PDL, um den Formalismus zu klären. Die Darstellung von Syntax und Semantik beruht im Wesentlichen auf den Erläuterungen in [CGL98], [GL96], [CGL95], [Gia95] und [GL94]. Eine Beschreibung des Entscheidungsverfahrens für $CPDL_g$ folgt im Unterabschnitt 3.3.4.

3.3.3.1 Propositional Dynamic Logic (PDL)

3.3.3.1.1 Einführung in PDL In der Mitte der 1960er Jahre wurden von THIELE[11] und ENGELER[12] unabhängig voneinander sogenannte *Programmlogiken* eingeführt, um die Eigenschaften von Programmen auf einer abstrakten, formalen Ebene beschreiben zu können. Zu diesen Programmlogiken zählt auch *Dynamic Logic*, eingeführt von PRATT[13]. Auf dieser Grundlage wurde *Propositional Dynamic Logic* (PDL) in den 1970er Jahren von FISCHER UND LADNER entwickelt.

Dynamische Logiken unterscheiden sich von klassischen Logiken insbesondere darin, dass sich der Wahrheitswert einer Formel ϕ mit der Zeit ändern kann. Durch das Konstrukt des *Programms* können in PDL die Werte von Variablen und dadurch die Werte von Formeln geändert werden. Jede Kombination von Variablenbelegungen wird ein *Zustand* genannt. Ein Programm beschreibt, wie ein Zustand in einen anderen Zustand überführt wird. Eine Abfolge von Zuständen, die ausgehend von einem Initialzustand (Eingabezustand) durch ein Programm erzeugt wird, heißt *Spur* (*trace*) und entspricht der Berechnung eines Programms. Eine Spur kann unendlich sein oder nach endlich vielen Transitionen konvergieren und in einem Ausgabezustand halten. In PDL werden Programme somit insbesondere als Eingabe-/Ausgaberelationen aufgefasst.

Definition 3.12 (Syntax von $\mathbf{CPDL_g}$). *In $CPDL_g$ gibt es zwei Sorten von Ausdrücken:* Programme *$(p, q, r, \ldots)$ und* Formeln *$(\phi, \psi, \ldots)$. Für jede Sorte sei eine abzählbar große Menge von Symbolen vorhanden: Prog als Menge der atomaren Programme und Prop als die Menge der atomaren Formeln. Weiterhin gibt es Konstruktoren, die aus einfacheren Ausdrücken komplexere aufbauen (es seien $A \in Prop; p \in Prog$; ϕ, ϕ_1, ϕ_2 Formeln; r_1, r_2 Programme; $k \in \mathbb{N}$):*

- *Programme:* $r ::= \quad p \mid r_1; r_2 \mid r_1 \cup r_2 \mid r^* \mid \phi? \mid r^-$
- *Formeln:* $\phi ::= \quad A \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle r \rangle \phi \mid [p]_{\leq k}\phi \mid [p^-]_{\leq k}\phi$
- *Vereinfachende Notationsformen:*

$$\begin{aligned} [r]\phi &:= \neg\langle r\rangle\neg\phi \\ \phi_1 \vee \phi_2 &:= \neg(\neg\phi_1 \wedge \neg\phi_2)) \\ \top &:= \phi \vee \neg\phi \\ \bot &:= \phi \wedge \neg\phi \\ \phi_1 \rightarrow \phi_2 &:= \neg(\phi_1 \wedge \neg\phi_2) \end{aligned}$$

[11] H. Thiele. Wissenschaftstheoretische Untersuchungen in Algorithmischen Sprachen. Theorie der Graphschemata-Kalküle. VEB Deutscher Verlag der Wissenschaften, Berlin 1966. Angabe nach [KT90].

[12] E. Engeler. Algorithmic properties of structures. Math. Systems Theory 1: 183–195, 1967. Angabe nach [KT90].

[13] V. R. Pratt. Semantical considerations on Floyd-Hoare-Logic. In: Proc. 17th Ann. IEEE Symp. on Foundations of Computer Sciences: 109–121, 1976. Angabe nach [KT90].

Die intuitive Bedeutung dieser Konstruktoren lässt sich folgendermaßen wiedergeben:

$[r]\phi$	Wenn Programm r hält, so gilt im Endzustand ϕ.
$\langle r\rangle\phi$	Es ist möglich, r auszuführen, so dass im Endzustand ϕ gilt.
$[p]_{\leq k}\phi$	Höchstens k Ausführungen von p halten in einem Zustand mit ϕ.
$r_1; r_2$	Führe erst r_1 aus, dann r_2.
$r_1 \cup r_2$	Wähle zufällig entweder r_1 oder r_2 und führe es aus.
r^*	Führe r zufällig oft hintereinander aus.
$\phi?$	Falls ϕ gilt, fahre fort, sonst Abbruch (Test).
p^-	Führe p rückwärts aus (*converse*).

Der Name $CPDL_g$ leitet sich aus der Erweiterung der einfachen PDL um folgende Konstruktoren ab:

- Der *converse*-Operator $^-$ wird im Namenskürzel durch das vorgestellte C bezeichnet.
- Konstrukte der Form $[p^-]_{\leq k}\phi$ nennt man *graded modalities*, für sie steht der Index $_g$.

Definition 3.13 (Semantik von $\mathbf{CPDL_g}$). *Die formale Semantik von PDL ist der Modallogik entlehnt. Diese beruht auf dem Begriff der* Kripke-Struktur. *Eine Kripke-Struktur $\mathcal{M}$ ist ein Paar $\mathcal{M} = (\mathcal{S}^{\mathcal{M}}, \cdot^{\mathcal{M}})$, wobei die beiden Komponenten wie folgt definiert sind:*

- $\mathcal{S}^{\mathcal{M}} \subseteq$ *Prop: Menge von Zuständen*
- $\cdot^{\mathcal{M}}$*: Eine Interpretationsfunktion, die jeder atomaren Formel ϕ eine Teilmenge $\phi^{\mathcal{M}} \subseteq S^{\mathcal{M}}$ (intuitiv: die Menge der Zustände, in denen ϕ gilt) und jedem atomaren Programm p eine binäre Relation $p^{\mathcal{M}} \subseteq S^{\mathcal{M}} \times S^{\mathcal{M}}$ (intuitiv: die Ein-/Ausgaberelation von p) zuordnet. Für komplexe Formeln und Programme wird deren Interpretation induktiv definiert:*

$A^{\mathcal{M}}$	$\subseteq S^{\mathcal{M}}$
$(\neg\phi)^{\mathcal{M}}$	$= S^{\mathcal{M}} \setminus \phi^{\mathcal{M}}$
$(\phi_1 \wedge \phi_2)^{\mathcal{M}}$	$= \phi_1^{\mathcal{M}} \wedge \phi_2^{\mathcal{M}}$
$(\langle r\rangle\phi)^{\mathcal{M}}$	$= \{s \mid \exists s'.(s, s') \in r^{\mathcal{M}} \wedge s' \in \phi^{\mathcal{M}}\}$
$([p]_{\leq k}\phi)^{\mathcal{M}}$	$= \{s \mid \sharp\{s' \mid (s, s') \in p^{\mathcal{M}} \wedge s' \in \phi^{\mathcal{M}}\} \leq k\}$
$([p^-]_{\leq k}\phi)^{\mathcal{M}}$	$= \{s \mid \sharp\{s' \mid (s', s) \in p^{\mathcal{M}} \wedge s' \in \phi^{\mathcal{M}}\} \leq k\}$
$(p)^{\mathcal{M}}$	$\subseteq S^{\mathcal{M}} \times S^{\mathcal{M}}$
$(r_1; r_2)^{\mathcal{M}}$	$= r_1^{\mathcal{M}} \circ r_2^{\mathcal{M}}$
$(r_1 \cup r_2)^{\mathcal{M}}$	$= r_1^{\mathcal{M}} \cup r_2^{\mathcal{M}}$
$(r^*)^{\mathcal{M}}$	$= (r^{\mathcal{M}})^* = \bigcup_{i\geq 0}(r^{\mathcal{M}})^i$
$(\phi?)^{\mathcal{M}}$	$= \{(s, s) \mid s \in \phi^{\mathcal{M}}\}$
$(r^-)^{\mathcal{M}}$	$= \{(s, s') \mid (s', s) \in r^{\mathcal{M}}\}$

Eine Formel ϕ gilt in Zustand s der Struktur M (notiert $M, s \models \phi$) gdw. $s \in \phi^M$. Eine Kripke-Struktur $\mathcal{M}$ heißt Kripke-Modell *(kurz: Modell) einer Formel ϕ, wenn es einen Zustand $s \in S^{\mathcal{M}}$ gibt, so dass $\mathcal{M}, s \models \phi$. Eine Formel ϕ heißt* erfüllbar*, falls es ein Modell von ϕ gibt, ansonsten heißt ϕ* unerfüllbar*.*

3.3.3.1.2 Entscheidungsverfahren für $\mathbf{CPDL_g}$ Für PDL-Dialekte gibt es zwei Verfahren zur Entscheidung des Erfüllbarkeitsproblems für Formeln.

(1) Tableau- oder Maximalmodell-Techniken
Diese Verfahren nutzen die *small (pseudo-) model property* der PDL in einem PDL-spezifischen *unwinding*-Prozess. Dieser erzeugt zu jeder PDL-Formel ein (unendliches) Modell höchstens exponentieller Größe. Die *small model property* besagt, dass wenn ϕ erfüllbar ist, ϕ in einem Zustand eines Modells mit nicht mehr als $2^{|\phi|}$ Zuständen erfüllt werden kann, wobei $|\phi|$ die Länge von (= die Anzahl von Symbolen in) ϕ ist. Ein naiver, nichtdeterministischer Algorithmus benötigt dann exponentielle Zeit. Für PDL gibt es jedoch auch einen deterministischen Algorithmus mit exponentieller Laufzeit und das Problem ist DEXPTIME-vollständig [KT90]. Dass die Erfüllbarkeit von Formeln in PDL EXPTIME-hart ist, ist schon von Fischer und Ladner gezeigt worden [FL79].

(2) Automatenbasierte Techniken
Hierbei profitiert man von der *tree model property*, welche die meisten PDL-Dialekte besitzen: Jedes Modell der Formel ϕ kann als ein markierter Graph betrachtet werden. Dieser Graph kann in einen Graph mit einer Baumstruktur beschränkten Verzweigungsgrads transformiert werden. Anschließend wird das Erfüllbarkeitsproblem für die PDL-Formel auf ein *emptyness*-Problem für endliche Automaten über derartige Bäume reduziert.

Für das Erfüllbarkeitsproblem für Formeln in $CPDL_g$ (und in *deterministic* $C_\Delta PDL$, das ist $CPDL$ mit lokaler Funktionalität der (inversen) atomaren Programme und einem `loop`-Konstruktor Δ) lässt sich ein Verfahren nach Methode (2) angeben und damit der Nachweis erbringen, dass das Erfüllbarkeitsproblem EXPTIME-vollständig ist [KT90, VW86, CGL95, Gia95]. Eine vereinfachte Darstellung dieses Verfahrens folgt in Abschnitt 3.3.4.

Eine umfassendere Einführung in PDL und eine Darstellung der Charakteristiken dieser Logiken bieten Kozen und Tiuryn in [KT90], $C_\Delta PDL$ wird von Streett [Str81] eingeführt, der auch ein EXPTIME-Entscheidungsverfahren für diese Logik angibt. Eine ausführliche Beschreibung der automatentheoretischen Techniken zur Entscheidung der Erfüllbarkeit von Formeln in *(converse) deterministic PDL (with looping)* bieten Vardi und Wolper [VW86].

3.3.3.2 Objektorientierte Modelle und PDL

Jedes Erfüllbarkeitsproblem für eine $\mathcal{DLR}_{reg}$- Wissensbasis kann auf ein Erfüllbarkeitsproblem für eine $CPDL_g$-Formel reduziert werden [CGL98, GL96, CGL95, Gia95, GL94].

Der Ansatz zu dieser Reduktion basiert auf der Beobachtung, dass zwischen den interpretierenden Strukturen von objektorientierten Modellen und Stellen-Transitionssytemen („*labeled transition systems*“) starke Ähnlichkeiten bestehen. Dazu stellt man das objektorientierte Modell als einen Graphen dar. In diesem Graphen werden Objekte als Knoten und Verknüpfungen/Verweise zwischen Objekten als gerichtete Kanten dargestellt. Die Knoten erhalten die Klassen, zu denen das repräsentierte Objekt gehört, als Markierungen; die Kanten werden jeweils mit dem Namen der entsprechenden Verknüpfung markiert.

Diese Darstellung kann auch als ein Transitionssystem aufgefasst werden. In diesem Transitionssystem entsprechen die Knoten jeweils Zuständen, die Knotenmarkierungen werden als Eigenschaften der Zustände aufgefasst. Die gerichteten Kanten im objektorientierten Modell entsprechen jeweils möglichen Zustandstransitionen und die Kantenmarkierungen den Namen der Transitionen.

Zur Beschreibung der Dynamik von Stellen-Transitionssystemen werden modale Logiken verwendet. Entsprechend der Analogie lassen sich diese Logiken auch zur Beschreibung objektorientierter Modelle verwenden. Ein brauchbares Pendant zu $\mathcal{DLR}_{reg}$ stellt dabei die zuvor beschriebene Logik $CPDL_g$ dar.

Diese Methode wird von Calvanese et al. in [CGL95] angewandt, um damit die Reduzierbarkeit von Schemata in einem objektorientierten Modell namens $\mathcal{CVL}$ auf *deterministic* $C_\Delta PDL$ nachzuweisen. Weiterhin geben die Autoren an, dass ein EXPTIME-vollständiges Verfahren existiert, welches das automatentheoretische Entscheidungsverfahren von Vardi und Wolpert für PDL [VW86] um die Berücksichtigung des `loop`-Operators und lokalen Determinismus von (inversen) Programmen erweitert.

Dieses Vorgehen wird von $\mathcal{CVL}$ auf $\mathcal{DLR}_{reg}$ übertragen. Dabei lassen sich die Anzahlbeschränkungen für Relationen in $\mathcal{DLR}_{reg}$ jedoch nicht ohne weiteres auf $CPDL$ abbilden. Es bedarf dazu der Erweiterung der PDL um das Konstrukt der *graded modalities*.

Ausführlicher wird die Methode der Reduktion einer *Description Logic* auf eine PDL in [Gia95] anhand einer ganzen Familie von *Description Logics* und PDLs vorgestellt.

3.3.3.3 Reduktion der $\mathcal{DLR}_{reg}$-Konzepterfüllbarkeit auf die Erfüllbarkeit einer $CPDL_g$-Formel

$\mathcal{DLR}_{reg}$ ist nach den in den Unterabschnitten 3.3.1 (Behandlung von ID- und FD-Zusicherungen) und 3.3.2 (Behandlung von ABox-Zusicherungen) angeführten Ergebnissen hinreichend für die Darstellung des vollen Sprachumfangs von $\mathcal{DLR}_{reg,ifd}$.

Das ursprünglich zu lösende Problem der logischen Implikation in einer $\mathcal{DLR}_{reg,ifd}$-Wissensbasis wurde bisher auf das $\mathcal{DLR}_{reg}$-Konzepterfüllbarkeitsproblem „Gilt $K \not\models (C \sqsubseteq \bot)$?“ reduziert. Das dazu duale Problem „Gilt $\neg(K \models (C \sqsubseteq \bot))$?“ ist identisch mit dem Problem „Ist $K \cup \{(C \sqsubseteq \bot)\}$ unerfüllbar?“.

In diesem Schritt wird das Problem „Ist $K \cup \{(C \sqsubseteq \bot)\}$ unerfüllbar?“ auf ein äquivalentes Problem der Form „Ist die $CPDL_g$-Formel Φ unerfüllbar?“ reduziert. Wie man dem nachfolgenden Unterabschnitt 3.3.4 entnehmen kann, ist das Problem der Erfüllbarkeit einer $CPDL_g$-Formel entscheidbar. Somit ist auch das dazu duale Problem der Unerfüllbarkeit einer $CPDL_g$-Formel entscheidbar und hat dieselbe algorithmische Komplexität.

Während De Giacomo eine Reihe von Reduktionen des Erfüllbarkeitsproblems in verschiedenen Erweiterungen von $\mathcal{ALC}_{reg}$ auf entsprechende PDL-Erweiterungen zeigt [Gia95] und Calvanese Gleiches für $\mathcal{LT}^- = \mathcal{ALCOQILDV}\Delta^-$ leistet [Cal96], zeigen Calvanese et al. in [CGL98] für $\mathcal{DLR}_{reg}$ nicht ein Entscheidungsverfahren für das Konzeptkonsistenzproblem, sondern ein komplexeres Verfahren für die Entscheidung von bestimmten Klassen des *Query-Containment-Problem*. Dieses Problem besteht in der Entscheidung, ob eine Wissensbasis-Anfrage q' in einer Anfrage q bezüglich eines Schemas K enthalten ist, d.h. ob für alle Modelle von K die Anfrage q' als Antwort stets eine Teilmenge der Antwort auf q liefert ($K \models q \subseteq q'$). Das Lösungsverfahren für das $\mathcal{DLR}_{reg}$-*Query-Containment-Problem* erfordert eine aufwändige Konstruktion zur Modellierung der Anfragebedingungen, die für das Entscheidungsverfahren der Konzeptkonsistenz nicht erforderlich sind. Auf diese zur Darstellung des Entscheidungsverfahrens für die Konzeptkonsistenz nicht notwendigen Anteile aus [CGL98] wird hier verzichtet.

Die Reduktion des Unerfüllbarkeitsproblems für ein Konzept C unter Berücksichtigung einer Wissensbasis K mit leerer ABox auf ein Unerfüllbarkeitsproblem für eine $CPDL_g$-Formel Φ wird mit Hilfe einer Transformation $\sigma(\cdot)$ vorgenommen, einer Funktion, die $\mathcal{DLR}_{reg}$-Ausdrücke induktiv auf $CPDL_g$-Ausdrücke abbildet. Der besseren Lesbarkeit wegen werden im $CPDL_g$-Widerpart der Wissensbasis K für atomare Programme und atomare Formeln dieselben Symbole verwendet, wie in K für atomare Relationen und atomare Konzepte. Die genauen Abbildungsvorschriften für σ sind in Tabelle 3.4 dargestellt. Das Schema $K = \langle E|TBox|\emptyset\rangle$ mit $TBox = \{S_1, \ldots, S_m\}$ wird insgesamt wie folgt in eine $CPDL_g$-Formel Φ übersetzt:

Zuerst werden sämtliche n-ären Relationen aus K *reifiziert*, das heißt sie werden auf binäre Relationen zurückgeführt. Die *Reifikation* einer n-ären Relation P bedeutet, dass Tupel von P durch neu einzuführende Individuen dargestellt werden; somit steht ein Individuum o_{t_i} für ein Tupel $t_i = (x_1, \ldots, x_n) \in P^{\mathcal{I}}$ der Extension der Relation P. Die Menge dieser Individuen repräsentiert die Menge der Tupel der ursprünglichen Relation. Die Individuen, welche die n Komponenten des Tupels bilden, werden über n neue Funktionen $f_1, \ldots, f_n$ identifiziert, wobei jeweils f_k das neue Individuum o_{t_i} auf die k-te Komponente des Tupels t_i abbildet, also $f_k(o_{t_i}) = x_k$ für $1 \leq k \leq n$. Da jedes Tupel jeder Relation einen eigenen Repräsentanten hat,

können für alle reifizierten Relationen dieselben f_i genutzt werden. Die Reifikation kann mit in $|K|$ polynomiellem Zeitaufwand durchgeführt werden.

Probleme, die sich daraus ergeben, dass es in einem Modell der reifizierten Wissensbasis im Allgemeinen mehrere Individuen geben kann, die ein und dasselbe Tupel kodieren, können vernachlässigt werden, da man auf Grundlage jedes Modells, in dem Tupel mehrfach kodiert sind, in polynomieller Zeit ein Modell konstruieren kann, in dem jedem Tupel genau ein Individuum entspricht.

Führt man für die Individuen, welche die Tupel einer Relation R repräsentieren, ein Konzept C_R ein, so kann man beispielsweise jede Rolleninklusionsabhängigkeit (RIC) direkt vermittels einer Klasseninklusionsabhängigkeit (CIC) ausdrücken:

$R_1 \sqsubseteq R_2$ ist äquivalent zu $C_{R_1} \sqsubseteq C_{R_2}$.

Von dieser Möglichkeit macht die Transformation σ Gebrauch, in der zu jeder atomaren $\mathcal{DLR}_{reg}$-Relation P_i ein Symbol P_i für eine die Relation repräsentierende atomare $CPDL_g$-Formel eingeführt wird.

$\mathcal{DLR}_{reg}$	$CPDL_g$
$\sigma(\top_n)$	$\top_n$
$\sigma(P)$	P
$\sigma((i/n : C))$	$T_n \wedge [f_i]\sigma(C)$
$\sigma(\neg R)$	$\top_n \wedge \neg\sigma(R)$
$\sigma(R_1 \sqcap R_2)$	$\sigma(R_1) \wedge \sigma(R_2)$
$\sigma(\epsilon)$	$\sigma(\top_1)?$
$\sigma(R\|_{[i][j]})$	$f_i^-; \sigma(R)?; f_j$
$\sigma(E_1 \circ E_2)$	$\sigma(E_1); \sigma(E_2)$
$\sigma(E_1 \sqcup E_2)$	$\sigma(E_1 \cup \sigma(E_2)$
$\sigma(E^*)$	$\sigma(E^*)$
$\sigma(\top_1)$	$\top_1$
$\sigma(A)$	A
$\sigma(\neg C)$	$\top_1 \wedge \neg\sigma(C)$
$\sigma(C_1 \sqcap C_2)$	$\sigma(C_1) \wedge \sigma(C_2)$
$\sigma(\exists E.C)$	$\langle\sigma(E)\rangle\sigma(C)$
$\sigma(\exists[i]R)$	$\langle f_i^-\rangle\sigma(R)$
$\sigma((\leq k[i]R))$	$[f_i^-]_{\leq k}\sigma(R)$
$\sigma(C_1 \sqsubseteq C_2)$	$[U](\sigma(C_1) \rightarrow \sigma(C_2))$
$\sigma(R_1 \sqsubseteq R_2)$	$[U](\sigma(R_1) \rightarrow \sigma(R_2))$

Tabelle 3.4: Induktive Translation $\sigma(\cdot)$ von Ausdrücken in $\mathcal{DLR}_{reg}$ nach $CPDL_g$ (erweitert nach [CGL98]).

Die Relationen in K seien bereits reifiziert, d.h. die $f_1, \ldots, f_{n_{max}}$ seien hiermit als atomare Programme eingeführt und jede ursprüngliche Relation P sei durch eine atomare Formel mit demselben Symbol P vertreten.

Die *universale Erreichbarkeitsrelation* U sei wie folgt definiert:

$$U := (f_1 \cup \cdots \cup f_{n_{max}} \cup f_1^- \cup \cdots \cup f_{n_{max}}^-)^*,$$

wobei n_{max} die höchste Stelligkeit einer vorkommenden Relation in K ist und die f_i die den neuen Funktionen $f_1, \ldots, f_{n_{max}}$ aus der Reifikation entsprechenden atomaren Programme sind.

Nun wird Φ wie folgt gebildet:

$$\Phi = \Phi_K \wedge \sigma(C \sqsubseteq \bot).$$

Dabei ist die Formel Φ_K der $CPDL_g$-Widerpart zur $\mathcal{DLR}_{reg}$-Wissensbasis K, der wie folgt konstruiert wird:

$$\begin{aligned}
\Phi_K \quad = \quad & \bigwedge_{1 \leq i \leq m} \sigma(S_i) \\
& \wedge [U](\top_1 \vee \cdots \vee T_{n_{max}}) \\
& \wedge [U]([f_i]_{\leq 1} \top) \text{ für jedes } i \in \{1, \ldots, n_{max}\} \\
& \wedge [U]([f_i]\bot \rightarrow [f_{i+1}]\bot) \text{ für jedes } i \in \{1, \ldots, n_{max}\} \\
& \wedge [U](\top_n \equiv \langle f_1 \rangle \top_1 \wedge \cdots \wedge \langle f_n \rangle \top_1 \wedge [f_{n+1}]\bot) \text{ für jedes } n \in \{2, \ldots, n_{max}\} \\
& \wedge [U](P \rightarrow \top_n) \text{ für jede atomare Relation } P \text{ der Stelligkeit } n \\
& \wedge [U](A \rightarrow \top_1) \text{ für jedes atomare Konzept A,}
\end{aligned}$$

wobei $n_{max}, f_1, \ldots, f_{n_{max}}$ wie oben definiert und $\top_1, \ldots \top_{n_{max}}$ atomare $CPDL_g$-Formeln seien, die den `TOP-CONCEPT`- und `TOP-RELATION`-Symbolen in $\mathcal{DLR}_{reg}$ entsprechen.

Offensichtlich ist die Bildung von $\Phi = \sigma(K)$ mit in $|K|$ polynomiellem Zeitaufwand möglich.

Das Problem „Ist $K \cup \{(C \sqsubseteq \top)\}$ unerfüllbar?“ kann nun als Unerfüllbarkeitsproblem für die $CPDL_g$-Formel Φ formuliert werden.

Nun löst man das Erfüllbarkeitsproblem in $CPDL_g$ für Φ. Ist Φ nicht erfüllbar, so gilt $K \not\models (C \sqsubseteq \top)$ und das Konzept C ist in K erfüllbar.

3.3.4 Entscheidungsverfahren für $CPDL_g$

Wie bereits erwähnt, basieren Entscheidungsverfahren für die Erfüllbarkeit von PDL-Formeln auf Tableau- oder Maximalmodell-Techniken und erlauben es im Zusammenhang mit einer *small model property* der PDL oder einer *small pseudo-model property* der PDL und einem PDL-spezifischen *unwinding*-Prozess, aus einer PDL-Formel ein (unendliches) Modell höchstens exponentieller Größe (bezogen auf die Länge der PDL-Formel) zu generieren. Leider muss jedoch für jede einzelne PDL der spezifische unwinding-Prozess gefunden und bewiesen werden [VW86].

Eine Alternative ist die Ausnutzung sogenannter *tree model properties*. Diese Eigenschaften modaler Logiken besagen, dass jedes Modell einer Struktur dieser Sprache

als ein markierter Graph betrachtet werden kann, der mit polynomiellem Aufwand in ein äquivalentes Modell transformiert werden kann, welches eine Baumstruktur mit beschränktem Verzweigungsgrad hat. Von dieser Darstellungsform profitiert das Verfahren, indem es das Erfüllbarkeitsproblem für die PDL-Formel zunächst mit linearem Aufwand in ein äquivalentes Erfüllbarkeitsproblem einer korrespondierenden $APDL$ übersetzt.

$APDL$ sind Varianten von PDL, in denen Formeln genauso wie in der PDL, Programme dagegen nicht durch reguläre Ausdrücke, sondern vielmehr durch *endliche Automaten* beschrieben werden. Weist die PDL ein `loop`/Δ-Konstrukt auf, wird dieses durch einen *Büchi-Automaten* ersetzt. Ein Büchi-Automat ist ein endlicher Automat, der ein unendliches Wort genau dann akzeptiert, wenn in der Folge der durchlaufenen Zustände auf einer Eingabe mindestens ein akzeptierender Zustand unendlich oft auftritt. Büchi-Automaten erkennen die Klasse der ω-regulären Sprachen. Diese sind unter den Bool'schen Operationen und der Verkettung von endlichen mit unendlichen Worten abgeschlossen und entsprechen insofern den regulären Ausdrücken. Es gilt:

- Ist p ein sequentieller Automat über einem Alphabet Σ, wobei Σ eine endliche Menge von atomaren Formeln und Tests ($\phi?$) ist, so ist p ein Programm.
- Ein Wort $w \in \Sigma^*$, das von einem Programm p akzeptiert wird, heißt *Ausführungssequenz von p.*
- Eine `loop`-Formel $\Delta(p)$ wird durch eine Formel $\Delta(B)$ ersetzt, wobei B ein sequentieller Büchi-Automat über einem Alphabet Σ von atomaren Programmen und Tests ist.
- Für die Semantik in $APDL$ gilt:
  - $p^{\mathcal{M}} = \{(u, u')|$ es gibt ein Wort $w := w_1 \cdots w_n,$ das von r akzeptiert wird, und Zustände $u_0, u_1, \ldots, u_n \in S^{\mathcal{M}},$ so dass $u = u_0, u' = u_n$ und für alle $i \in \{1, \ldots, n\}$ gilt $(u_{i-1}, u_i) \in (w_i)^{\mathcal{M}}\}$.
  - $\mathcal{M}, u \models \Delta(B)$ gdw. ein unendliches Wort $w := w_1 w_2 \cdots$ von B akzeptiert wird und eine unendliche Sequenz $u_0, u_1, \ldots$ von Zuständen in $S^{\mathcal{M}}$ existiert, so dass $u_0 = u$ und $(u_{i-1}, u_i) \in (w_i)^{\mathcal{M}}$ für alle $i \geq 1$.

Die *tree model property* der PDL ermöglicht es, sich bei der Suche nach Modellen auf *tree models* zu beschränken. Dies geschieht, indem zur Eingabe ϕ für das Erfüllbarkeitsproblem in $APDL$ ein endlicher Automat konstruiert wird, der solche Eingaben akzeptiert, die sogenannte Hintikka-Bäume zur Formel ϕ darstellen.

Hintikka-Bäume sind n-äre Bäume über einem Alphabet Σ, das aus allen Teilmengen von $CL^+(\phi) \cup X$ besteht. Dabei ist $CL^+(\phi)$ eine PDL-spezifische Erweiterung des Fischer-Ladner-Abschlusses $CL(\phi)$; X ist eine Menge von Symbolen, die es dem

Automaten ermöglicht, zusätzliche Bedingungen zu überprüfen, welche die Korrespondenz der Hintikka-Bäume zu den *tree models* von ϕ herstellen. Der Verzweigungsgrad der Hintikka-Bäume ist höchstens $\sharp(CL^+(\phi))$ und damit polynomiell in $|\phi|$.

Die genaue Konstruktion des endlichen Automaten hängt insbesondere vom Sprachumfang der PDL ab. Insbesondere *coverse*, lokaler Determinismus und der `loop`-Operator verursachen Wechselwirkungen, die nur durch komplizierte Konstruktionen, wie Hybride Baumautomaten, aufgelöst werden können. Insgesamt ist die Anzahl der Zustände des Automaten exponentiell in $|\phi|$. Die Formel ϕ ist genau dann erfüllbar, wenn die vom Automaten erkannte Sprache nicht leer ist. Der Aufwand zur Lösung dieses Problems kann durch weitere Verfeinerungen des Automaten so verringert werden, dass sich für die Entscheidung der Erfüllbarkeit von Formeln der PDL-Variante $C_\Delta PDL_g$ insgesamt ein deterministischer exponentieller Zeitaufwand in $|\phi|$ ergibt.

Genauere Betrachtungen zur Vorgehensweise kann man [VW86] und [Cal96] entnehmen.

Häufig ist es auch möglich, komplexere PDL-Varianten auf einfachere zurückzuführen. So lässt sich das Erfüllbarkeitsproblem für Formeln in $CPDL_g$ durch eine polynomielle Reduktion auf das Erfüllbarkeitsproblem in $CPDL$ zurückführen und damit der Nachweis erbringen, dass auch dieses Erfüllbarkeitsproblem EXPTIME-vollständig ist. De Giacomo zeigt dazu eine polynomielle Reduktion der Erfüllbarkeit in der PDL $\mathcal{DIN}$ (entspricht $CPDL_g$) auf die Erfüllbarkeit in der PDL $\mathcal{DIF}$ (entspricht *deterministic* $CPDL$) und schließlich in der PDL $\mathcal{DI}$ (entspricht $CPDL$) [Gia95].

Als Ergebnis bleibt hier festzustellen, dass für die Erfüllbarkeitsprobleme für Formeln in PDL, CPDL (converse-PDL), CPDL$_g$ (converse-PDL mit *graded modalities*) und $C_\Delta PDL$ (converse-PDL mit `loop`) mittels der vorgestellten Methoden DEXPTIME-Vollständigkeit nachgewiesen wurde [KT90, VW86, CGL95, Cal96].

3.3.5 Gesamtkomplexität des Entscheidungsverfahrens

Die Gesamtkomplexität des Entscheidungsverfahrens für Konzepterfüllbarkeit, Enthaltenseinsproblem und logische Implikation ist für sämtliche $\mathcal{DLR}$-Varianten und die meisten anderen ausdrucksstarken *Description Logic*-Sprachen DEXPTIME-vollständig.

Der Aufwand ergibt sich aus dem zu Grunde liegenden DEXPTIME-vollständigen Entscheidungsverfahren für die Erfüllbarkeit von PDL-Formeln und den (ggf. iterierten) polynomiellen Reduktionschritten von der Ausgangs-DL bis zur entsprechenden PDL.

3.4 Vorstellung der DL DLClass

Die Description Logic DLClass wird von KHIZDER ET AL. in [KTW01] beschrieben. Hier erfolgt eine freie Wiedergabe der Definitionen des Originalartikels.

Definition 3.14 (Pfadausdruck). *Sei F eine Menge von Attributbezeichnern. Ein Pfadausdruck Pf wird definiert durch die Grammatik*

$$Pf ::= f.Pf \mid Id,$$

wobei $f \in F$.

Definition 3.15 (Konzeptbeschreibung). *Sei C ein primitives Konzept. Dann werden allgemeine Konzepte D durch folgende Grammatik beschrieben:*

$$\begin{array}{rcl} D & ::= & C \\ & \mid & (all\, f\; D) \\ & \mid & (fd\, C:\; Pf_1, \ldots, Pf_k \rightarrow Pf), k > 0 \\ & \mid & (and\, D\; D) \end{array}$$

Definition 3.16 (Konzeptinklusionszusicherung). *Sei C eine primitives und D eine allgemeines Konzept. Dann ist eine Konzeptinklusionszusicherung ein Ausdruck der Form $C < D$. Eine Terminologie ist eine endliche Menge von Konzeptinklusionszusicherungen.*

Konzeptinklusionszusicherungen in DLClass sind also immer primitive Konzeptbeschreibungen.

Definition 3.17 (Interpretation). *Eine Interpretation ist eine Struktur $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$. Dabei ist Δ eine Domäne von Objekten und $.^{\mathcal{I}}$ eine Interpretationsfunktion, die jedem primitiven Konzept C eine Teilmenge $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ der Domäne zuordnet und jedem primitiven Attribut eine totale Funktion über die Domäne $f : \Delta^{\mathcal{I}} \rightarrow \Delta^{\mathcal{I}}$. Die Interpretationsfunktion wird auf Pfadausdrücke erweitert durch $Id^{\mathcal{I}} = \lambda x.x$ und $(f.Pf)^{\mathcal{I}} = Pf^{\mathcal{I}} \circ f^{\mathcal{I}}$. Die Interpretationsfunktion wird wie folgt auf allgemeine Konzepte erweitert:*

$$\begin{array}{rcl} (all\, f\; D)^{\mathcal{I}} & = & \{o \in \Delta^{\mathcal{I}} | f^{\mathcal{I}}(o) \in D^{\mathcal{I}}\} \\ (fd\, C:\; Pf_1, \ldots, Pf_k \rightarrow Pf)^{\mathcal{I}} & = & \{o \in \Delta^{\mathcal{I}} | \forall o' \in C^{\mathcal{I}}: \\ & & \bigwedge_{i=1}^{k} Pf_i^{\mathcal{I}}(o) = Pf_i^{\mathcal{I}}(o') \Rightarrow Pf^{\mathcal{I}}(o) = Pf^{\mathcal{I}}(o')\} \\ (and\, D_1\; D_2)^{\mathcal{I}} & = & D_1^{\mathcal{I}} \cap D_2^{\mathcal{I}} \end{array}$$

Definition 3.18 (Modell). *Eine Interpretation $\mathcal{I}$ heißt Modell, wenn sie alle Konzeptinklusionszusicherungen erfüllt, d.h. für jede Konzeptinklusionszusicherung $C < D$ gilt $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.*

In [TW01] wird `DLClass` variiert und dann unter den Bezeichnungen $\mathcal{DLF}$, $\mathcal{DLFR}$ und $\mathcal{DLFD}$ gefasst. Die geänderten Konstruktorsätze sind in Tabelle 3.5 wiedergegeben. Konzeptinklusionszusicherungen haben hier die Form $D \sqsubseteq E$. Eine Interpretation ist hier eine Struktur $(\Delta^{\mathcal{I}}, \leq, .^{\mathcal{I}})$, wobei $\Delta^{\mathcal{I}}$ eine Domäne von Objekten, $\leq$ eine partielle oder lineare Ordnung auf $\Delta^{\mathcal{I}}$ und $.^{\mathcal{I}}$ eine Interpretationsfunktion ist, die zusätzlich binären Rollen Teilmengen von $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ zuordnet. Inverse Rollen werden wie folgt interpretiert: $(P^{-1})^{\mathcal{I}} = \{(y,x) | (x,y) \in (P)^{\mathcal{I}}\}$.

Syntax		C	F	FR	FD	Semantik (Definition von $.^{\mathcal{I}}$)
$D ::=$	C	x	x	x	x	$C^{\mathcal{I}} \subset \Delta^{\mathcal{I}}$
$\mid$	$D_1 \sqcap D_2$	x	x	x	x	$(D_1)^{\mathcal{I}} \cap (D_2)^{\mathcal{I}}$
$\mid$	$\forall f.D$	x	x	x	x	$\{x \mid (f)^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}\}$
$\mid$	$\neg D$	o	x	x	x	$\Delta^{\mathcal{I}} \setminus (D)^{\mathcal{I}}$
$\mid$	$\forall R.D$	o	o	x	o	$\{x \mid \forall y \in \Delta^{\mathcal{I}}.(x,y) \in (R)^{\mathcal{I}} \Rightarrow y \in (D)^{\mathcal{I}}\}$
$\mid$	$\exists R.D$	o	o	x	o	$\{x \mid \exists y \in \Delta^{\mathcal{I}}.(x,y) \in (R)^{\mathcal{I}} \wedge y \in (D)^{\mathcal{I}}\}$
$E ::=$	D	o	x	x	x	
$\mid$	$E_1 \sqcap E_2$	o	o	o	x	$(E_1)^{\mathcal{I}} \cap (E_2)^{\mathcal{I}}$
$\mid$	$\forall f.E$	o	o	o	x	$\{x \mid (f)^{\mathcal{I}}(x) \in (E)^{\mathcal{I}}\}$
$\mid$	$D\{\mathrm{Pf}_1^{\sim_1}, \ldots, \mathrm{Pf}_k^{\sim_k}\} \rightarrow \mathrm{Pf}^{\sim}$	x†	o	o	x	$\{x \mid \forall y \in D^{\mathcal{I}}$: $\bigwedge_{i=1}^{k} (\mathrm{Pf}_i)^{\mathcal{I}}(x) \sim_i (\mathrm{Pf}_i)^{\mathcal{I}}(y) \Rightarrow (\mathrm{Pf})^{\mathcal{I}}(x) \sim (\mathrm{Pf})^{\mathcal{I}}(y)\}$

Tabelle 3.5: Syntax und Semantik von $\mathcal{DLF}$ und Erweiterungen

Anmerkungen zu Tabelle 3.5: Die Tabelle gibt den jeweiligen Konstruktorumfang von `DLClass` (C), $\mathcal{DLF}$ (F), $\mathcal{DLFR}$ (FR) und $\mathcal{DLFD}$ (FD) an.
Dabei ist C ein primitives Konzept, f eine totale Funktion $f : \Delta \longrightarrow \Delta$, $k > 0$ eine ganze Zahl, $\sim \in \{<, \leq, =, \geq, >\}$ ein Vergleichoperator, R sei eine primitive binäre Rolle P oder deren Inverse P^{-1}.
† In `DLClass` sind in diesem Konstruktor lediglich atomare anstelle beliebiger Konzepte und ausschließlich „=“ als Vergleichs-/Ordnungsoperatoren $\sim_1, \ldots, \sim_k, \sim$ erlaubt.

Satz 3.2 (Logische Implikation in `DLClass`, [KTW01] Korollar 10). *Die logische Implikation in `DLClass` ist DEXPTIME-vollständig.*

Beweis: *Polynomielle Reduktion des $Datalog_{nS}$-* recognition problem *über die logische Implikation in DLFD (eine als Teilmenge in `DLClass` enthaltene* Description Logic*-Sprache) auf die logische Implikation in `DLClass` und schließlich wieder auf das* recognition problem *in $Datalog_{nS}$:*

$$Datalog_{nS} \rightarrow DLFD \rightarrow \texttt{DLClass} \rightarrow Datalog_{nS}$$

■

Satz 3.3 (Logische Implikation in $\mathcal{DLF}$, [TW01] Korollar 12). *Die logische Implikation in $\mathcal{DLF}$ ist DEXPTIME-vollständig.*

Beweis: *Polynomielle Reduktion des* recognition problem *für $Datalog_{nS}$ auf die logische Implikation in $\mathcal{DLF}$ und umgekehrt.* ∎

Satz 3.4 (Logische Implikation in $\mathcal{DLFR}$, [TW01] Theorem 14). *Die logische Implikation in $\mathcal{DLFR}$ ist DEXPTIME-vollständig.*

Beweis: *Rollen und inverse Rollen werden in $\mathcal{DLF}$ durch Attribute modelliert. Das Konzeptkonsistenzproblem für $\mathcal{DLFR}$ ist dann durch ein Unerfüllbarkeitsproblem für $\mathcal{DLF}$ zu lösen.* ∎

Satz 3.5 (Logische Implikation in $\mathcal{DLFD}$, [TW01] Theorem 19). *Die logische Implikation in $\mathcal{DLFD}$ ist DEXPTIME-vollständig.*

Beweis: *Beliebige Implikationsprobleme für $\mathcal{DLFD}$ können auf (mehrere) sogenannte einfache Implikationsprobleme reduziert werden. Ein $\mathcal{DLFD}$ Implikationsproblem $\mathcal{T} \models \mathcal{C}$ heißt* einfach*, wenn jede Konzeptinklusionszusicherung in $\mathcal{T}$ entweder die Form $D_1 \sqsubseteq D_2$ oder $D_1 \sqsubseteq D_2\{Pf_1^{\sim 1}, \ldots, Pf_k^{\sim k}\} \rightarrow Pf^{\sim}$ hat. Einfache Implikationsprobleme können auf Unerfüllbarkeitsprobleme für $\mathcal{DLF}$ reduziert werden.* ∎

Bemerkung 3.1 (Erweiterungen von $\mathcal{DLFR}$). Toman und Weddell merken weiterhin an, dass *guarded equality- and order-generating dependencies* auch in $\mathcal{DLFR}$ eingeführt werden können. Dies beruht darauf, dass die (durch Attribute modellierten) (inversen) Rollen nicht mit den *dependencies* interagieren – die funktionalen Attribute in den Pfadausdrücken und die Attribute für die Modellierung der Rollen bleiben strikt getrennt. Die dergestalt erweiterte Sprache heiße $\mathcal{DLFRD}$

Weiterhin deuten Toman und Weddell an, dass die Erweiterung von $\mathcal{DLFR}$ um Anzahlbeschränkungen für Rollen, Rollenkonstruktoren, Rolleninklusionszusicherungen, Rollenhierarchien etc. möglich sei [TW01].

3.5 Vorstellung der DL $\mathcal{ALCQI}_{reg}$

Die *Description Logic* $\mathcal{ALCQI}_{reg}$ wird von Calvanese et al. in [CGNL99] ausführlich beschrieben. Dort werden auch die Reduktionsschritte (*internalization of the TBox*, *reification of relations*) zu den Entscheidbarkeitsbeweisen für diese Sprache detailliert dargestellt. Gleiches gilt für [CG03], wo Calvanese und De Giacomo eine ausführliche Darstellung der Beweise zur Entscheidbarkeit des Erfüllbarkeitsproblems in $\mathcal{ALCQI}_{reg}$ und der dabei angewandten Methoden bieten, dabei jedoch auch das Folgern in Wissensbasen mit ABoxes berücksichtigen.

Der Name $\mathcal{ALCQI}_{reg}$ entstammt einer systematischen Notation für die Dialekte der *Description Logic*-Sprachfamilie $\mathcal{AL}$. Die Grundsprache $\mathcal{AL}$ umfasst folgenden Konstruktorsatz: $C_1 \sqcap C_2, \top, \bot, \neg A, \forall R.C, \exists R.\top$, also den Durchschnitt allgemeiner Konzepte, das allgemeinste und das leere Konzept, das Komplement atomarer Konzepte,

die Werterestriktion für Rollen und die unqualifizierte existentielle Quantifikation über Rollen. Erweiterungen dieses Sprachumfangs werden durch Buchstaben angezeigt:

- $\mathcal{U}$: $C_1 \sqcup C_2$, *union of concepts*,
- $\mathcal{E}$: $\exists R.C$, *full/qualified existential quantification*,
- $\mathcal{C}$: $\neg C$, *complement of arbitrary concepts* (entspricht $\mathcal{UE}$),
- $\mathcal{N}$: $(\leq nR), (\geq nR)$, *(unqualified) number restrictions*,
- $\mathcal{R}$: $R_1 \sqcap R_2$, *intersection of roles*,
- $\mathcal{I}$: R^-, *inverse roles*,
- $\mathcal{Q}$: $(\exists^{\geq} nQ.C), (\exists^{\leq} nQ.C)$, *qualified number restrictions*,
- $\mathcal{O}$: $\{o\}$, *individuals as concepts* (*nominals*),
- $.^{reg}$: $R^*, R_1 \sqcup R_2, R_1 \circ R_2, \mathrm{id}(C), R^-$, *regular expressions over roles.*

Dabei steht A für atomare Konzepte, C für beliebige Konzepte, R für beliebige Rollen, Q für sogenannte *basic roles*, das sind atomare Rollen P oder (so *inverse roles* vorhanden sind) deren Inverse P^-, und n für eine natürliche Zahl.

Definition 3.19 ($\mathcal{ALCQI}^{reg}$- Wissensbasis). *Eine $\mathcal{ALCQI}^{reg}$-Wissensbasis*

$$\mathcal{K} = \langle \mathcal{T}_k, \mathcal{A}_k \rangle$$

besteht aus einer TBox $\mathcal{T}_K$, die Konzeptinklusionszusicherungen der Form $C_1 \sqsubseteq C_2$ enthält, sowie einer ABox $\mathcal{A}_K$, die Instanzzusicherungen der Form $C(a)$ oder $P(a_1, a_2)$ enthält. Dabei sind a, a_1, a_2 Symbole für Individuen, P ist eine atomare Rolle und C, C_1, C_2 sind beliebige Konzepte.

Es handelt sich somit um eine freie TBox.

Definition 3.20 (Interpretation). *Eine* Interpretation $\mathcal{I}$ *ist ein Paar* $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, *wobei* $\Delta^{\mathcal{I}}$ *eine* Interpretationsdomäne *von Individuen und* $.^{\mathcal{I}}$ *eine* Interpretationsfunktion *ist, die Konzeptausdrücken Mengen von Individuen und Rollen Mengen von Individuenpaaren zuordnet. Insbesondere ordnet* $.^{\mathcal{I}}$

- *jedem atomaren Konzept A eine Teilmenge von $\Delta^{\mathcal{I}}$,*
- *jeder atomaren Rolle P eine Teilmenge von $(\Delta^{\mathcal{I}})^2$ und*
- *jedem Individuenbezeichner a ein Individuum aus $\Delta^{\mathcal{I}}$ zu.*

Komplexe, mittels Konstruktoren gebildete, Ausdrücke werden gemäß Tabelle 3.6 interpretiert.

Definition 3.21 (Syntax und Semantik von $\mathcal{ALCQI}_{reg}$). *Syntax und Semantik für Ausdrücke in dieser Sprache sind in Tabelle 3.6 dargestellt. Eine Interpretation $\mathcal{I}$ heißt* Modell *einer $\mathcal{ALCQI}_{reg}$-Wissensbasis, wenn jede Zusicherung in $\mathcal{T}_K \cup \mathcal{A}_K$ von $\mathcal{I}$ erfüllt wird. Eine Zusicherung ist erfüllt:*

- $C_1 \sqsubseteq C_2$ *gdw.* $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$,
- $C(a)$ *gdw.* $a^{\mathcal{I}} \in C^{\mathcal{I}}$,
- $P(a_1, a_2)$ *gdw.* $(a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in P^{\mathcal{I}}$).

$\mathcal{K}$ heißt erfüllbar*, wenn es ein Modell von $\mathcal{K}$ gibt.*

Satz 3.6 (Komplexität der logischen Implikation in $\mathcal{ALCQI}_{reg}$, [CG03] Theorem 5.18). *Das Problem der (unbeschränkten) logischen Implikation über $\mathcal{ALCQI}_{reg}$-Wissensbasen ist EXPTIME-vollständig.*

Beweis: *$\mathcal{ALCQI}_{reg}$-Wissensbasen können in ein einzelnes Konzept* internalisiert *werden. Das Implikationsproblem kann somit linear auf das Konzepterfüllbarkeitsproblem reduziert werden. Der Beweis zur Entscheidbarkeit und Komplexität des Konzepterfüllbarkeitsproblems beruht auf einer Kette polynomieller Reduktionen der internalisierten $\mathcal{ALCQI}_{reg}$-Wissensbasis. Zuerst werden qualifizierte Anzahlbeschränkungen durch (unqualifizierte) funktionale Beschränkungen modelliert, anschließend werden auch diese eliminiert. Zuletzt werden inverse Rollen eliminiert, so dass nunmehr ein äquivalentes Problem für die Konzepterfüllbarkeit in $\mathcal{ALC}_{reg}$ vorliegt.*

$$\mathcal{ALCQI}_{reg} \rightarrow \mathcal{ALCFI}_{reg} \rightarrow \mathcal{ALCI}_{reg} \rightarrow \mathcal{ALC}_{reg}$$

Konzepterfüllbarkeitsprobleme für $\mathcal{ALC}_{reg}$ können polynomiell auf Erfüllbarkeitsprobleme für PDL reduziert werden, die bekanntermaßen EXPTIME-vollständige sind. ■

3.6 Semantische Bedingungen in DL

Semantische Bedingungen können in *Description Logics* entweder als Zusicherungen formuliert werden, denen Modelle genügen müssen (Beispiel: ID- und FD-Zusicherungen in $\mathcal{DLR}_{ifd}$; allgemein in DL: Konzeptinklusionszusicherungen in der Terminologie), oder sie werden eigens als Konstruktoren eingeführt und dienen dann der Beschreibung von Konzepten, die der entsprechenden Bedingung genügen (Beispiel: `fd`-Konstruktor in `DLClass`).

Syntax		Semantik (Definition von $.^{\mathcal{I}}$)
$C ::=$	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$\mid$	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$\mid$	$C_1 \sqcap C_2$	$(C_1)^{\mathcal{I}} \cap (C_2)^{\mathcal{I}}$
$\mid$	$C_1 \sqcup C_2$	$(C_1)^{\mathcal{I}} \cup (C_2)^{\mathcal{I}}$
$\mid$	$\forall R.C$	$\{o \in \Delta^{\mathcal{I}} \mid \forall o' : (o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
$\mid$	$\exists R.C$	$\{o \in \Delta^{\mathcal{I}} \mid \exists o' : (o, o') \in R \wedge o' \in C^{\mathcal{I}}\}$
$\mid$	$\exists^{\geq n} Q.C$	$\{o \in \Delta^{\mathcal{I}} \mid \sharp\{o' \mid (o, o') \in Q^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \geq n\}$
$\mid$	$\exists^{\leq n} Q.C$	$\{o \in \Delta^{\mathcal{I}} \mid \sharp\{o' \mid (o, o') \in Q^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \leq n\}$
$Q ::=$	P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
$\mid$	P^-	$\{(o, o') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (o', o) \in P^{\mathcal{I}}\}$
$R ::=$	P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
$\mid$	R^-	$\{(o, o') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (o', o) \in R^{\mathcal{I}}\}$
$\mid$	$R_1 \sqcup R_2$	$(R_1)^{\mathcal{I}} \cup (R_2)^{\mathcal{I}}$
$\mid$	$R_1 \circ R_2$	$(R_1)^{\mathcal{I}} \circ (R_2)^{\mathcal{I}}$
$\mid$	R^*	$(R^{\mathcal{I}})^*$
$\mid$	$\mathrm{id}(C)$	$\{(o, o) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid o \in C^{\mathcal{I}}\}$
$^\dagger\mid$	$P_1 \sqcap P_2$	$(P_1)^{\mathcal{I}} \cap (P_2)^{\mathcal{I}}$
$^\dagger\mid$	$P_1^- \sqcap P_2^-$	$(P_1^-)^{\mathcal{I}} \cap (P_2^-)^{\mathcal{I}}$
$^\dagger\mid$	$P_1 \setminus P_2$	$(P_1)^{\mathcal{I}} \setminus (P_2)^{\mathcal{I}}$
$^\dagger\mid$	$P_1^- \setminus P_2^-$	$(P_1^-)^{\mathcal{I}} \setminus (P_2^-)^{\mathcal{I}}$
$^\dagger\mid$	$(S_1 \subseteq S_2)$	$\{o \in \Delta^{\mathcal{I}} \mid \{o' \mid (o, o') \in S_1^{\mathcal{I}}\} \subseteq \{o' \mid (o, o') \in S_2^{\mathcal{I}}\}\}$
$^\dagger S ::=$	Q	s.o.
$^\dagger\mid$	$S_1 \sqcup S_2$	s.o.
$^\dagger\mid$	$S_1 \setminus S_2$	s.o. (in S_1, S_2 dürfen insgesamt entweder keine Inversen atomarer Rollen oder ausschließlich Inverse atomarer Rollen auftreten)

Tabelle 3.6: Syntax und Semantik von $\mathcal{ALCQI}_{reg}$

Anmerkungen zu Tabelle 3.6: A ist ein atomares Konzept, P eine atomare Rolle; n ist eine positive, ganze Zahl.

† Diese Zeilen stellen mögliche, in DEXPTIME entscheidbare Erweiterungen von $\mathcal{ALCQI}_{reg}$ dar, vgl. [CGNL99, Abschnitte 6.1 und 6.2].

3.6.1 Uniqueness Constraints

In den hier vorgestellten *Description Logic*-Sprachen werden unterschiedliche Arten von Uniqueness Constraints unterstützt, beispielsweise werden sie in $\mathcal{DLR}_{ifd}$ in Gestalt von ID-Zusicherungen (einer Art Schlüsselbedingungen) für Konzepte und FD-Zusicherungen für Rollen (funktionale Abhängigkeiten) eingeführt. Während in $\mathcal{DLR}_{ifd}$ PFDs nicht unterstützt werden (Ansätze zu einer entsprechenden Erweiterung werden in Kapitel 5 diskutiert), wird in `DLClass` und $\mathcal{DLFD}$ eigens ein Konzeptkonstruktor dafür eingeführt.

3.6.1.1 Pfadabhängigkeiten

Bemerkung 3.2. Da man jede Pfadabhängigkeit $(\mathrm{Pf}_1, \ldots, \mathrm{Pf}_n \rightarrow \mathrm{Pf}'_1, \ldots, \mathrm{Pf}'_k)$ leicht auf k Pfadabhängigkeiten der Form $(\mathrm{Pf}_1, \ldots, \mathrm{Pf}_n \rightarrow \mathrm{Pf}'_i), 1 \leq i \leq k$ reduzieren kann, wird im Weiteren nur noch diese einfachere Form verwendet.

PFDs werden in der *Description Logic*-Sprache `DLClass` mit Hilfe eines Konzeptkonstruktors

$$(\textbf{fd}\ C:\ \mathrm{Pf}_1, \ldots, \mathrm{Pf}_n \rightarrow \mathrm{Pf}),\ k > 0$$

— dabei ist C ein Konzept, Pf und die Pf_i sind Pfadausdrücke — eingeführt. Der Konstruktor wird wie folgt interpretiert:

$$\left\{ o \in \Delta^{\mathcal{I}} | \forall o' \in C^{\mathcal{I}} : \quad \left(\bigwedge_{i=1}^{n} \mathrm{Pf}_i^{\mathcal{I}}(o) = \mathrm{Pf}_i^{\mathcal{I}}(o') \right\} \right) \Rightarrow \mathrm{Pf}^{\mathcal{I}}(o) = \mathrm{Pf}^{\mathcal{I}}(o')\ \}$$

Dabei werden alle Attribute als totale Funktionen f mit $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow \Delta^{\mathcal{I}}$ aufgefasst, Pfadausdrücke werden definiert durch $\mathrm{Pf} ::= f.\mathrm{Pf} | \mathrm{Id}$ und für die Interpretation von Pfadausdrücken gilt $\mathrm{Id}^{\mathcal{I}} = \lambda x.x$, $(f.\mathrm{Pf})^{\mathcal{I}} = \mathrm{Pf}^{\mathcal{I}} \circ f^{\mathcal{I}}$.

Die *Description Logic*-Sprachen $\mathcal{DLR}_{reg,ifd}$ und $\mathcal{ALCQI}_{reg}$ halten keine besonderen Konstrukte für PFDs vor. Folglich müssten PFDs unter Verwendung der vorhandenen Konstruktoren modelliert werden. Überlegungen dazu finden sich in Kapitel 5.

3.6.1.2 Funktionale Abhängigkeiten

Bemerkung 3.3. Da man jede funktionale Abhängigkeit

$$(A_1, \ldots, A_n \longrightarrow A'_1, \ldots, A'_k)$$

leicht auf k funktionale Abhängigkeiten der Form

$$(A_1, \ldots, A_n \longrightarrow A'_i), 1 \leq i \leq k$$

reduzieren kann, wird im Weiteren nur noch diese einfachere Form verwendet.

In $\mathcal{DLR}_{reg,ifd}$ werden funktionale Abhängigkeiten für Rollen durch *FD-Zusicherungen* der folgenden Form innerhalb von $\mathcal{S}_K$ eingeführt:

$$(\textbf{fd}\ R:\ i_1, \ldots, i_h \rightarrow i)$$

Diese Zusicherung wird wie folgt interpretiert (vgl. Def. 3.10): „Zwei Tupel von R, die auf den Komponenten $i_1, \ldots, i_h$ übereinstimmen, stimmen auch in Komponente i überein.“

Bemerkung 3.4. Unäre funktionale Abhängigkeiten führen zur Unentscheidbarkeit von $\mathcal{DLR}_{reg,ifd}$. Der Beweis nach [CGL01] ist direkt von $\mathcal{DLR}_{ifd}$ auf $\mathcal{DLR}_{reg,ifd}$ übertragbar. Allerdings sind unäre Schlüssel für Relationen modellierbar: Durch die Konzeptinklusionszusicherung

$$\top_1 \sqsubseteq (\leq 1\ [i]R)$$

wird ausgedrückt, dass Komponente i der Relation R einen Schlüssel für R darstellt. Somit können auch unäre funktionale Abhängigkeiten für binäre Relationen R beschrieben werden: Nämlich

$$(\text{fd}\ R:\ 1 \longrightarrow 2)$$

durch

$$\top_1 \sqsubseteq (\leq 1\ [1]R).$$

Nicht modelliert werden können dagegen unäre funktionale Abhängigkeiten im Zusammenhang mit nicht-binären Relationen.

In `DLClass` und $\mathcal{DLFD}$ können funktionale Abhängigkeiten durch den `fd`-Konstruktor bzw. den Konstruktor $D\{\mathrm{Pf}_1^{\sim_1}, \ldots, \mathrm{Pf}_k^{\sim_k}\} \rightarrow \mathrm{Pf}^{\sim}$ jeweils mit Pfaden der Länge 1 ausgedrückt werden.

In $\mathcal{ALCQI}_{reg}$ können zwar mittels $\top \sqsubseteq \exists^{\leq 1} R.\top$ unäre funktionale Abhängigkeiten bzw. unäre Schlüssel für Rollen ausgedrückt werden, die direkte Formulierung von komplexeren funktionalen Abhängigkeiten ist jedoch nicht möglich.

3.6.1.3 Schlüssel

In $\mathcal{DLR}_{reg,ifd}$ werden Schlüsselbedingungen in Form von *ID-Zusicherungen* als Zusicherungen

$$(\textbf{id}\ C:\ [i_1]R_1, \ldots, [i_h]R_h)$$

für Konzepte eingeführt.

Identitätszusicherungen werden wie folgt interpretiert (vgl. Def. 3.10): Zwei Instanzen von C können nicht gemeinsam an den Rollen $R_1, \ldots, R_h$ mittels deren Komponenten $i_1, \ldots, i_h$ beteiligt sein, d.h. wenn je 2 Tupel der Rollen R_k bis auf die Komponenten $[i_k]R_k$ übereinstimmen und die Tupel lediglich durch Vertauschen von a und b entstehen, so sind a und b identisch.

Als Folge dieser Interpretation können in $\mathcal{DLR}_{reg,ifd}$ auch solche Attribute Bestandteile eines Schlüssels sein, die optional oder mehrwertig sind.

Weiterhin können, wie bereits oben bemerkt, durch Konzeptinklusionszusicherungen der Art

$$\top_1 \sqsubseteq (\leq 1\ [i]R)$$

unäre Schlüssel für beliebige Relationen modelliert werden.

In `DLClass` können Schlüssel über ein Konzept C leicht durch eine PFD ausgedrückt werden:

$$C < (\text{fd } C :\ \text{Pf}_1, \ldots, \text{Pf}_n \rightarrow \text{Id}),$$

wobei die Menge der Pfade $\text{Pf}_1, \ldots, \text{Pf}_n$ dem Schlüssel entspricht. Entsprechendes gilt für $\mathcal{DLFD}$.

In $\mathcal{ALCQI}_{reg}$ können unäre Schlüssel für Relationen wie in $\mathcal{DLR}_{reg,ifd}$ ausgedrückt werden:

$$\top \sqsubseteq \exists^{\leq 1} R.\top.$$

Die Konstruktion von *identification assertions* nach dem Vorbild von $\mathcal{DLR}_{reg,ifd}$ (vgl. Abschnitt 3.3.1) zum Ausdruck nicht-unärer Schlüssel ist jedoch denkbar.

3.6.2 Onto-Abhängigkeiten

Keine der drei vorgestellten *Description Logic*-Sprachen hält ein eigenes Sprachelement zum Ausdruck dieser Art von semantischen Bedingungen vor. Sie müssen daher durch andere Sprachkonzepte umschrieben werden. Die Darstellung von Onto-Abhängigkeiten verlangt Konstruktoren wie die Bildung der inversen Rolle. Weiteres zur Formulierung dieser Abhängigkeiten entnehme man Kapitel 5, hier insbesondere dem Abschnitt 5.7.

3.6.3 Inklusionsabhängigkeiten

Eine *Konzeptinklusionsabhängigkeit* (*class subsumption constraint*) besagt, dass alle Individuen, die einem ersten Konzept entsprechen, ebenfalls einem zweiten Konzept entsprechen. Eine *Relationsinklusionsabhängigkeit* (*relation subsumption constraint*) besagt, dass alle Tupel, die Element einer ersten Relation sind, ebenfalls Elemente einer zweiten Relation sind.

Seien $C_1, C_2 \in \mathcal{C}_D$ Konzepte, $R_1, R_2 \in \mathcal{R}_D$ Relationen. Eine Konzeptinklusionsabhängigkeit ($C_1 \subseteq C_2$)/Relationsinklusionsabhängigkeit ($R_1 \subseteq R_2$) kann man wie folgt interpretieren:

$$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}} \qquad \text{bzw.} \qquad R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}.$$

Somit entspricht der Konzept-/Relationsinklusionsabhängigkeit in $\mathcal{DLR}_{reg,ifd}$ einfach folgende Konzept-/Relationsinklusionszusicherung in der TBox:

$$(C_1 \sqsubseteq C_2) \qquad \text{bzw.} \qquad (R_1 \sqsubseteq R_2).$$

In `DLClass` lässt sich die Konzeptinklusionsabhängigkeit durch

$$C_1 < C_2$$

ausdrücken, wobei C_1 ein primitives Konzept sein muss. In den Erweiterungen von DLClass, wie $\mathcal{DLF}$, $\mathcal{DLFD}$ oder $\mathcal{DLFD}$, wird diese Beschränkung relativiert (vgl. Abschnitt 3.4). Allerdings ist der Ausdruck von Relationsinklusionszusicherungen nicht möglich.

Auch in $\mathcal{ALCQI}_{reg}$ kann eine Konzeptinklusionsabhängigkeit

$$C_1 \subseteq C_2 \qquad \text{durch} \qquad C_1 \sqsubseteq C_2$$

ausgedrückt werden. Rolleninklusionsabhängigkeiten können dagegen nicht ausgedrückt werden.

Kapitel 4

Stand der Forschung

Dieses Kapitel bietet eine Übersicht über bekannte Ergebnisse zur Entscheidbarkeit der logischer Implikation von semantischen Bedingungen in verschiedenen Datenmodellen. Von besonderem Interesse sind dabei Aussagen zur Komplexität solcher Entscheidungsalgorithmen sowie Aussagen zur Entscheidbarkeit unterschiedlicher Kombinationen von Konstruktoren in *Description Logics*.

4.1 Relationales Datenmodell

Da zum einen das relationale Datenmodell (RDM) als Spezialfall des OODM aufgefasst werden kann [Sch01, Bis95b], zum anderen auch *Description Logic*-Sprachen im relationalen Datenmodell modelliert werden können, sind die Ergebnisse zum Implikationsproblem im relationalen Datenmodell mit unterschiedlichen Typen von semantischen Bedingungen auch für die hier behandelte Fragestellung relevant.

Implikationsprobleme für semantische Bedingungen im relationalen Datenmodell sind im Allgemeinen gut verstanden: Sie sind entscheidbar, solange in den den semantischen Bedingungen entsprechenden prädikatenlogischen Formeln erster Ordnung keine existentiell gebundene Variable positiv auftritt. Ansonsten (eine existentiell gebundene Variablem tritt positiv auf) ist das Implikationsproblem unentscheidbar, da solch eine Variable in den Inferenzprozeduren die Erzeugung unendlich vieler Terme verursachen kann[Bis95a].

Durch die Möglichkeit der Reduktion des Wortproblems für (endliche) Monoide auf (endliche) Implikationsprobleme für Inklusions- und funktionale Abhängigkeiten ist bekannt, dass das Implikationsproblem im relationalen Datenbankmodell in Verbindung mit *allgemeinen Inklusionsabhängigkeiten* und *allgemeinen funktionalen Abhängigkeiten* unentscheidbar ist [Mit83, CV85]. Dies gilt bereits für binäre Inklusionsabhängigkeiten. Beschränkt man sich dagegen auf *unäre Inklusionsabhängigkeiten*, so sind sowohl unbeschränkte, als auch endliche Implikation in polynomieller Zeit entscheidbar [KCV83]. Da sich die Unentscheidbarkeit aus den Wechselwirkungen der funktionalen Abhängigkeiten und Inklusionsabhängigkeiten ergibt,

beschäftigen sich einige Untersuchungen mit der Fragestellung, wie man diese Wechselwirkungen durch bestimmte Anforderungen an die Mengen von semantischen Bedingungen vermeiden kann [LL99, LL01]. Insbesondere werden dabei Beschränkungen auf Mengen von azyklischen und baumartigen Inklusionsabhängigkeiten diskutiert.

4.2 F-Logic

Eine Variante der *Frame Logic*, oder kurz *F-Logic*, wurde bereits in Kapitel 2 vorgestellt. Die Originalvariante von *F-Logic* nach [KLW95] umfasst zusätzliche Merkmale, wie etwa mengenwertige Attribute. Die in Kapitel 2 vorgestellte Variante bietet insofern nur einen Ausschnitt der *F-Logic*. Kifer et al. zeigten in [KLW95] die Vollständigkeit und Korrektheit einer auf Resolution beruhenden Beweisprozedur für ihre *F-Logic*.

Die Variante nach Biskup und Polle [BP03] bietet, wie in Kapitel 2 gezeigt, eine vollständige und korrekte Axiomatisierung der allgemeinen Implikation für folgende Klassen von semantischen Bedingungen:

- *Klasseninklusionsabhängigkeiten* (CICs) sind unäre Inklusionsabhängigkeiten.
- *Onto-Abhängigkeiten* (OCs) sind Erreichbarkeitsbedingungen bzw. Aussagen über inverse Attribute. Diese Attribute müssen funktional sein und dürfen als „eigentliche“ Attribute für nur genau eine Klasse definiert sein.
- *Pfadabhängigkeiten* (PFDs) über wohldefinierte Ketten von Attributen, die sowohl auf der linken, wie auch auf der rechten Seite mindestens eine Pfadfunktion aufweisen.

Die Entscheidbarkeit für diese Menge von Abhängigkeiten bleibt jedoch offen.

4.3 Graphbasierte Datenmodelle

Pfadabhängigkeiten (*path functional dependencies*, PFDs) werden von Weddell 1992 im Zusammenhang eines semantischen, graphbasierten Datenmodells eingeführt [Wed92].

Ein Datenbankschema besteht dabei aus einer endlichen Menge von Klassenschemata der Form $C\{P_1 : C_1, \ldots, P_n : C_n\}$. Die P_i heißen Eigenschaften (*properties*) von C. Dabei handelt es sich um Funktionen, die Instanzen der Klasse C auf Instanzen der Klassen C_i abbilden. Der Bildbereich (*range*) einer Eigenschaft P_i der Klasse C, $Ran(C, P_i)$, ist die Klasse C_i.

Die Domäne einer Eigenschaft P, $Dom(P)$, ist die Menge aller Klassen C, in deren Klassenschemata P als Eigenschaft auftritt. $Props(C)$ ist die Menge aller Eigenschaftsbezeichner P_i im Klassenschema von Klasse C. Die Menge aller Klassen im Datenbankschema S heißt $Classes(S)$.

Eine Interpretation des Schemas ist ein gerichteter Graph $G = (V, A)$ wobei V eine Menge von mit Klassenbezeichnern markierten Knoten und A eine Menge von mit Eigenschaftenbezeichnern markierten, gerichteten Kanten zwischen den Knoten aus V darstellt. Die Markierung eines Knoten v sei $l_{Cl}(v)$. Die Interpretation G muss dann drei Bedingungen genügen:

(1) *Property value integrity*: Wenn $(u \xrightarrow{P} v) \in A$, dann $l_{Cl}(u) \in Dom(P)$ und $l_{Cl}(v) = Ran(l_{Cl}(u), P)$.

(2) *Property functionality*: Wenn $\{(u \xrightarrow{P} v), (u \xrightarrow{P} w)\} \subseteq A$, dann $v = w$.

(3) *Property value completeness*: Wenn $u \in V$, dann gibt es für alle $P \in Props(l_{Cl}(u))$ ein $(u \xrightarrow{P} v) \in A$.

Die Menge der wohlgeformten Pfadfunktionen über einem Datenbankschema S, $PF_{Wf}(S)$, ist die kleinste Menge, so dass gilt:

(1) $\mathrm{Id} \in PF_{Wf}(S)$, wobei $Dom(\mathrm{Id}) := Classes(S)$ und $Ran(C, \mathrm{Id}) := C$ für alle $C \in Classes(S)$.

(2) Wenn $pf, P \in PF_{Wf}(S), C \in Dom(pf)$ und $P \in Props(Ran(C, pf))$, dann $pf \circ P \in PF_{Wf}(S)$, wobei
$Dom(pf \circ P) := \{C_1 \in Dom(pf) | P \in Props(Ran(C_1, pf))\}$ und
$Ran(C_1, pf \circ P) := Ran(Ran(C_1, pf), P)$ für alle $C_1 \in Dom(pf \circ P)$.

PFDs haben die Form $C(pf_1 \cdots pf_m \rightarrow pf_{m+1} \cdots pf_n)$, für $1 \leq m < n$. PFDs sind echte Verallgemeinerungen von funktionalen Abhängigkeiten. Eine funktionale Abhängigkeit entspricht gerade einer PFD, in der sämtliche Pfadfunktionen lediglich die Länge 1 haben.

Eine Schlüssel-PFD (*key-PFD*) hat die Form $C(pf_1 \cdots pf_m \rightarrow \mathrm{Id})$. Schlüssel-PFDs sind echte Verallgemeinerungen von gewöhnlichen Schlüsselbedingungen (deren Pfadfunktionen wiederum lediglich die Länge 1 haben).

Ein Datenbankschema S heißt *azyklisch* genau dann, wenn $PF_{Wf}(S)$ endlich ist.

Weddell gibt für dieses Datenmodell eine korrekte und vollständige Axiomatisierung der generellen logischen Implikation von PFDs an. Für Spezialfälle (ausschließlich Schlüssel-PFDs als semantische Bedingungen oder beliebige PFDs in einem azyklischen Schema) wird auch eine Entscheidungsprozedur angegeben.

In [IW94] betrachten Ito und Weddell im Kontext desselben Datenmodells auch PFDs mit leerer linker Seite (also ohne Prämissen: $C(\quad \rightarrow pf_1 \cdots pf_n)$, für $n \geq 1$). Für unendliche Datenbanken gilt dieselbe Axiomatisierung wie in [Wed92] und das generelle Implikationsproblem für die so erweiterte Klasse von PFDs ist entscheidbar. Im endlichen Fall gilt diese Axiomatisierung nicht.

Nach Angabe der Autoren lassen sich diese Ergebnisse einfach auf ein um *class is-a* Zusicherungen (d.h. unäre Klasseninklusionsabhängigkeiten) erweitertes Datenmodell mit PFDs übertragen. Das Ergebnis ist eine korrekte und vollständige Axiomatisierung für beliebige PFDs in unendlichen Datenbanken. Für endliche Datenbanken sind die Axiome unvollständig. Das generelle Implikationsproblem für beliebige PFDs ist entscheidbar. Für Schlüssel-PFDs existiert sogar ein effizienter Polynomialzeitalgorithmus.

Dasselbe Datenmodell wird von van Bommel und Weddell in [vBW94] im Zusammenhang mit *path equations* (PEs, das sind Gleichheitszusicherungen über *feature paths*) und Pfadabhängigkeiten (PFDs) untersucht. Erlaubt man ausschließlich PEs als semantische Bedingungen, so ist die logische Implikation in beliebigen Schemata unentscheidbar. Erlaubt man zusätzlich zu PEs auch PFDs, so kann man eine korrekte und vollständige Axiomatisierung für die generelle Implikation angeben. Für stratifizierte Schemata (in diesen ist die Länge der Pfade in PEs beschränkt), in denen alle PFDs auch Schlüssel sind, wird Entscheidbarkeit in exponentieller *worst case*-Laufzeit festgestellt.

In [BFW03] stellen Buneman et al. Ergebnisse für das Implikationsproblem von *Inklusionsabhängigkeiten*, *inversen Relationen* und *Pfadabhängigkeiten* im Zusammenhang mit verschiedenen, baumbasierten Datenmodellen (semistrukturiert, objektorientiert (mit Records, Klassen, rekursiven Strukturen) ohne Mengen, objektorientiert mit Mengen, objektorientiert mit endlichen Mengen) vor. Dabei ist die (endliche und generelle) Implikation nur im objektorientierten Modell mit Klassen, Records und rekursiven Strukturen, jedoch ohne Mengen, in kubischer Zeit entscheidbar und ansonsten unentscheidbar.

4.4 Description Logics

Bislang wurden vielfältige Arten von *Description Logic* (DL)-Sprachen untersucht, die sich insbesondere in den Kombinationen verfügbarer Rollen- und Konzeptkonstruktoren unterscheiden. Diese Kombinationen bestimmen maßgeblich die Art der jeweils darstellbaren semantischen Bedingungen. Für das Implikationsproblem liegen dabei einige Erkenntnisse zur Interaktion bestimmter Konstruktorklassen und deren Auswirkungen auf die Entscheidbarkeit bzw. Komplexität der Inferenzalgorithmen vor. An dieser Stelle erfolgt eine Zusammenfassung von Erkenntnissen zu *Description Logic*-Sprachen und Konstruktoren, die einen Bezug zur vorliegenden

Aufgabenstellung, insbesondere zur Darstellung der fraglichen Klassen von semantischen Bedingungen haben. Eine Einführung in *Description Logics* mit Klärung der entsprechenden Termini erfolgte bereits in Kapitel 3.

4.4.1 Implikationsprobleme in Description Logics

Für die meisten *Description Logics* unterscheiden sich der endliche und der generelle Fall des Erfüllbarkeits- und Implikationsproblems. Soweit nicht anderes angemerkt wird, beziehen sich Angaben in diesem Abschnitt auf die generellen Fälle.

In allen *Description Logic*-Sprachen, die bezüglich der Konzepte unter Bool'schen Operationen abgeschlossen sind, sind die Probleme der logischen Implikation und der Konzept(un)erfüllbarkeit linear aufeinander reduzierbar. Deswegen entfällt bisweilen die explizite Erwähnung des einen oder anderen Problems.

4.4.2 Inklusionsabhängigkeiten

Getypte unäre *Inklusionsabhängigkeiten* (INDs) können in allen *Description Logic*-Sprachen formuliert und entschieden werden, insbesondere dienen unäre Inklusionsabhängigkeiten als TBox-Zusicherungen zur Beschreibung der Konzepte innerhalb der Terminologie (wie $C_1 \sqsubseteq C_2$).

Eine allgemeinere Form von TBox-Zusicherungen stellen die von Tobies in [Tob00] diskutierten *cardinality restriction axioms* für Konzepte dar. Solche Beschränkungen der Form $(\leq n\, C)$ oder $(\geq n\, C)$ fordern, dass das Konzept C mindestens (höchstens) n Instanzen hat. Dies ist die allgemeinste bislang diskutierte Form von TBox-Zusicherungen und umfasst u.a. auch Inklusionsabhängigkeiten der Form $C_1 \sqsubseteq C_2$. Die *Description Logic*-Sprachen $\mathcal{ALCQ}$ und $\mathcal{ALCQO}$ sind mit *cardinality restriction axioms* EXPTIME-vollständig, $\mathcal{ALCQI}$, $\mathcal{ALCQIO}$ sind sowohl bezüglich der Konzepterfüllbarkeit ohne TBox, bezüglich der Konzepterfüllbarkeit in Gegenwart einer freien TBox, als auch der Konzepterfüllbarkeit unter Berücksichtigung einer generalisierten TBox mit *cardinality restriction axioms* NEXPTIME-vollständig.

Weiterhin erlauben einige *Description Logic*-Sprachen Zusicherungen von Inklusionsabhängigkeiten auch für Rollen (binäre Inklusionsabhängigkeiten) bzw. über n-äre Relationen (so etwa in den $\mathcal{DLR}$-Dialekten). Derartige Inklusionsabhängigkeiten heißen dann *role inclusion axioms* (RIAs) oder *role inclusion constraints* (RICs). Eine Menge von RICs bezeichnet man dann als eine *Rollenhierarchie* oder *role box*. In Analogie zu den Konzeptinklusionszusicherungen und der Terminologie unterscheidet man auch unterschiedliche Arten von RICs und *role boxes* (vgl. Abschnitt 3.1.2.2.3, S. 27).

4.4.3 Spezielle Konstrukte für semantische Bedingungen

Einige *Description Logics* definieren eigens Konstruktoren oder Klassen von Zusicherungen, um semantische Bedingungen explizit ausdrücken zu können.

So führen *Khizder et al.* in [KTW01] für die *Description Logic* `DLClass` einen Konzeptkonstruktor $(\text{fd } C : \text{Pf}_1, \ldots, \text{Pf}_k \rightarrow \text{Pf}), k > 0$ ein, der ein Konzept liefert, für dessen Instanzen dieselbe formulierte Pfadabhängigkeit gilt, wie für Instanzen des Konzepts C. Die logische Implikation in `DLClass` ist EXPTIME-hart. Zu einer Einführung in `DLClass` und Erweiterungen dieser Logik sei auf Abschnitt 3.4 der vorliegenden Arbeit verwiesen.

Zum Ausdruck von Schlüsselbedingungen führen Calvanese et al. in [CGL00] sogenannte *key assertions* sowohl für Relationen $(\text{key } R\ \$i_i, \ldots, \$i_h)$ als auch für Konzepte $(\text{key } C\ [\$i_1]R_1, \ldots, [\$i_h]R_h)$ in die *Description Logic* $\mathcal{DLR}$ ein und beweisen, dass Erfüllbarkeit und logische Implikation in $\mathcal{DLR}_{key}$ EXPTIME-vollständig sind.

Dieselben Autoren setzen ihre Arbeit in [CGL01] fort und führen $\mathcal{DLR}_{ifd}$ als eine Erweiterung von $\mathcal{DLR}$ um *identification assertions* (ID-Zusicherungen) für Konzepte, $(\text{id } C\ [i_1]R_1, \ldots, [i_h]R_h)$, und *functional dependency assertions* (FD-Zusicherungen) für Relationen, $(\text{fd } R\ i_1, \ldots, i_h \rightarrow j), h \geq 2$, ein. Erfüllbarkeits- und Implikationsproblem der resultierenden *Description Logic* sind EXPTIME-vollständig. Werden zusätzlich unäre FD-Zusicherungen (mit $h = 1$) zugelassen, so wird das Problem der Wissensbasiserfüllbarkeit unentscheidbar. Eine detaillierte Darstellung von $\mathcal{DLR}_{ifd}$ findet sich in Abschnitt 3.2 der vorliegenden Arbeit.

4.4.4 Anzahlbeschränkungen

Description Logic-Sprachen mit Konstruktoren, die *funktionale Beschränkungen* oder *qualifizierte Anzahlbeschränkungen* für Konzepte und Rollen umfassen, beschränken die Rollen innerhalb dieser Konstruktorausdrücke in der Regel auf primitive Rollen und gegebenenfalls auf deren Inverse. Die Verwendung allgemeiner Rollenausdrücke (mit Komposition oder Vereinigung von Rollen) an dieser Stelle führt zur Unentscheidbarkeit der wichtigen Probleme für diese *Description Logic*-Sprachen.

Als Beispiel sei hier die Unentscheidbarkeit der *Description Logic*-Sprache $\mathcal{SHIN}$ genannt, wenn in den Anzahlbeschränkungen beliebige (insbesondere transitive) Rollen erlaubt werden ($\mathcal{SHIN}^+$). Die Unentscheidbarkeit wird durch die Reduktion eines Dominoproblems auf $\mathcal{SHIN}^+$ gezeigt [HST99].

Die Erweiterung von $\mathcal{ALC}$ um Anzahlbeschränkungen für Kompositionen von Rollen hat ein entscheidbares Erfüllbarkeits- und Enthaltenseinsproblem. Erlaubt man in den beschränkten Rollen jedoch Komposition und Vereinigung von Rollen sowie inverse Rollen, so sind beide Probleme unentscheidbar. Gleiches gilt, falls die beschränkten Rollen Komposition und Durchschnitt von Rollen enthalten dürfen. Für

$\mathcal{ALC}_{trans}$, die Erweiterung von $\mathcal{ALC}$ um transitive Rollen, ist bereits die Komposition von Rollen alleine innerhalb von Anzahlbeschränkungen unentscheidbar [BN03].

4.4.5 Komplement von Rollen

Description Logic-Sprachen mit (qualifizierten) Anzahlbeschränkungen sind in der Regel nicht mit einer *Komplementbildung über Rollen* ausgestattet. Eine solche erlaubt in Verbindung mit (qualifizierten) Anzahlbeschränkungen nämlich Festlegungen der Größe der Interpretationsdomäne, was einer Voraussetzung des Entscheidbarkeitsbeweises für das Erfüllbarkeitsproblem zuwiderläuft. Daher wird auf diese Kombination verzichtet und anstelle des Komplements die „Negation“ von Rollen als ihre Differenz zu einer `TOP-RELATION` $\top_n$ mit der Interpretation $P^{\mathcal{I}} \subseteq \top_n^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$ für alle primitiven Rollen P eingeführt, so etwa in den *Description Logic*-Sprachen in [Gia95] und den $\mathcal{DLR}$-Varianten. Somit ist eine Universalrelation (Δ^n) in diesen *Description Logic*-Sprachen keine *basic role* und damit als Rolle in Anzahlbeschränkungen nicht nutzbar.

Eine Bemerkung bei Hustadt et al. [HSG04] belegt, dass das Rollenkomplement insbesondere in Verbindung mit der Komposition von Rollen grundlegende Probleme verursacht:

> „From the literature on PDL-like modal logics and relation algebras it is known that role composition and role complement together with role intersection or union lead to undecidability.“

4.4.6 Role-value-maps (RVMs)

Role-value-maps (RVMs) sind Konzeptkonstruktoren der Form $(\mathbf{R_1} \subseteq \mathbf{R_2})$. $\mathbf{R_1}$ und $\mathbf{R_2}$ sind dabei Ketten von Rollen der Form $\mathbf{R_1} = R_1^1 \circ \ldots \circ R_1^m, \mathbf{R_2} = R_2^1 \circ \ldots \circ R_2^n$. Eine übliche Interpretation ist

$$(\mathbf{R_1} \subseteq \mathbf{R_2})^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} | \forall o' : (o, o') \in \mathbf{R_1}^{\mathcal{I}} \rightarrow (o, o') \in \mathbf{R_2}^{\mathcal{I}}\}.$$

Spezialfälle, die sogenannten *equality RVMs*, fordern in der Interpretation anstelle von „$\rightarrow$“ ein „$\leftrightarrow$“. Der allgemeine Fall wird dann *containment RVM* genannt.

RVMs stellen ausdrucksstarke Konstruktoren dar. Leider ist das Enthaltenseinsproblem (*subsumption problem*) in *Description Logic*-Sprachen mit RVMs unentscheidbar, selbst wenn die DL ansonsten ausschließlich den Durchschnitt von Konzepten und die Werte-Restriktion als weitere Konstruktoren umfasst [Sch89][1]. Selbst eine so einfache *Description Logic*-Sprache wie $\mathcal{LV}$, die um RVMs erweiterte Variante von $\mathcal{ALC}$, ist damit unentscheidbar [Cal96, Theorem 4.4.1].

[1] Angabe nach [BN03, HSG04].

Allgemeine RVMs führen zum Verlust der *tree model property* von *Description Logics*, was wiederum zu Problemen beim Folgern über Kompositionen von Rollen (Ketten von Rollen) führt. Einen Beweis für die Unentscheidbarkeit des Enthaltenseinsproblems führen NEBEL UND SMOLKA durch eine Reduktion des Wortproblems für eine Klasse von Thue-Systemen auf ein Enthaltenseinsproblem für $\mathcal{ALC}$ mit RVMs [NeS91]. Ein weiterer Beweis findet sich bei DONINI [Don03].

Werden die Rollen in RVMs auf Bool'sche Kombinationen von atomaren Rollen (oder deren Inverse) beschränkt, wird die Entscheidbarkeit nicht nachteilig beeinflusst. In der DL $\mathcal{LT}^-$ (entspricht der DL $\mathcal{CVL}$ aus [CGL95]) etwa werden daher RVMs auf *basic roles* (atomare Rollen, Vereinigung und Differenz von *basic roles*) und deren Inverse beschränkt [Cal96]. Gleiche Beschränkungen gelten hier auch für *qualified number restrictions* und die Differenz von Rollen (hier werden sogar Inverse *basic roles* ausgeschlossen).

Bestimmte RVMs lassen sich auch in *Description Logic*-Sprachen ausdrücken, die keinen eigenen Konstruktor dafür vorhalten. So ist beispielsweise der $\mathcal{DLR}$-Ausdruck $(\forall(R_1 \sqcap \neg R_2).\bot)$ äquivalent zum RVM-Ausdruck $(R_1 \subseteq R_2)$. In $\mathcal{CIQ}$ werden qualifizierte Anzahlbeschränkungen auf *basic roles* beschränkt, da bereits Ausdrücke über beliebige Rollen der Form $(\leq 1(R_1 \sqcup R_2).\top)$ eine Art RVM darstellen und zur Unendscheidbarkeit führen. Mit funktionalen Beschränkungen und der Vereinigung von Rollen lassen sich demnach einfache RVMs darstellen.

Werden die verketteten Rollen in RVMs auf *funktionale Rollen* (auch Attribute oder *features* genannt), das sind Rollen für die gilt $(a, b), (a, c) \in R^{\mathcal{I}} \rightarrow b = c$, beschränkt, so kann die Entscheidbarkeit erhalten werden.

Ähnlich den RVMs sind auch die Konstruktoren *feature-agreement* (auch *path-equation* oder `SAME-AS` genannt) und *feature-disagreement*. Sie sind für Ketten funktionaler Rollen $\mathbf{R_1}, \mathbf{R_2}$ definiert:

$$(\mathbf{R_1} \downarrow \mathbf{R_2})^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} | \exists o' : (o, o' \in \mathbf{R_1}^{\mathcal{I}}) \wedge (o, o' \in \mathbf{R_2}^{\mathcal{I}})\},$$

$$(\mathbf{R_1} \not\downarrow \mathbf{R_2})^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} | \exists o', o'' : o' \neq o'' \wedge (o, o' \in \mathbf{R_1}^{\mathcal{I}}) \wedge (o, o'' \in \mathbf{R_2}^{\mathcal{I}})\}.$$

Als `same-as`-Konstruktor sind *feature-agreements* etwa Bestandteil der *Description Logic*-Sprache `CLASSIC` [BBGR89].

Beschränkt man sich auf funktionale Rollen, so ist auch $\mathcal{ALC}$ mit *feature-agreements* und *-disagreements* entscheidbar [BN03]. Sobald jedoch zusätzlich *general inclusion axioms*, das sind Rolleninklusionszusicherungen (RIAs/RICs), der transitive Abschluss für funktionale Rollen oder zyklische Terminologien in die *Description Logic*-Sprache eingeführt werden, wird die *Description Logic* unentscheidbar [BN03].

4.4.7 Reguläre Ausdrücke

Als reguläre Ausdrücke bezeichnet man binäre Relationen, die durch die Anwendung von Konstruktoren zur Bildung regulärer Ausdrücke über Rollen gebildet werden.

Konstruktoren zur Bildung regulärer Ausdrücke sind insbesondere die Identitätsrelation id, die relationale Komposition $R_1 \circ R_2$, der reflexiv-transitive Abschluss R^*, die Disjunktion $R_1 \cup R_2$ und die Inversion R^- für Rollen. Reguläre Ausdrücke haben eine wichtige Funktion bei der Formulierung von komplexen Rollen, wie etwa *feature paths* in *Description Logics*.

Die Bildung *regulärer Ausdrücke* über Rollen erlaubt in Verbindung mit einem *well-founded*-Konstrukt und einer geeigneten Semantik die induktive Definition von Strukturen (Listen, Folgen) und die Formulierung von Bedingungen für dieselben. Das *well-founded*-Konstrukt ermöglicht dabei Zusicherungen über die Endlichkeit von induktiven Strukturen.

Kapitel 5

Ansätze zur Reduktion der F-Logic auf Description Logics

In diesem Kapitel werden Ansätze zur Reduktion des Implikationsproblems der in Kapitel 2 eingeführten *F-Logic* nach Biskup und Polle [BP03] auf Implikationsprobleme in den in Kapitel 3 eingeführten *Description Logics* vorgestellt. Dabei werden die Unzulänglichkeiten der angewandten *Description Logics* zur Lösung des Problems herausgestellt.

5.1 Prinzip: Modellierung der F-Logic in DL

Wie durch die Vorstellung von *Description Logics* in Kapitel 3 verdeutlicht wurde, stellen *Description Logics* einen ausdrucksstarken Formalismus zur Beschreibung objektorientierter Datenmodelle dar. Außerdem konnte für zahlreiche DL-Sprachen die Frage nach der Entscheidbarkeit von Konzepterfüllbarkeitsproblemen und logischer Implikation positiv beantwortet werden, so dass Entscheidungsalgorithmen zur Lösung dieser Probleme vorliegen. Andererseits gibt es zahlreiche DL-Sprachen, für die bezüglich der Entscheidbarkeit des Konzepterfüllbarkeitsproblems bzw. der logischen Implikation negative Ergebnisse veröffentlicht wurden.

Vor diesem Hintergrund erscheint es viel versprechend, *Description Logics* zur Reformalisierung der anstehenden Fragestellung zu benutzen und sich so der Frage nach der Entscheidbarkeit der logischen Implikation von Klasseninklusionsabhängigkeiten, Pfadabhängigkeiten und Onto-Abhängigkeiten in der vorgestellten *F-Logic* anzunähern.

Dazu bedient man sich des Prinzips der polynomiellen Reduktion des Entscheidungsproblems für die logische Implikation in der *F-Logic* auf das Entscheidungsproblem für die logische Implikation in einer korrespondierenden *Description Logic*. Es geht also im Folgenden darum, eine polynomiell zeitbeschränkte Funktion anzugeben, mit der eine Eingabe für das Entscheidungsproblem der logischen Implikation in

der *F-Logic* auf eine Eingabe für die logische Implikation in einer *Description Logic* abgebildet werden kann.

5.2 Zu übertragende Konstrukte

Bei der Reduktion des Implikationsproblems in *F-Logic* auf ein Implikationsproblem in einer *Description Logic* müssen alle wesentlichen Charakteristika des ursprünglichen Problems berücksichtigt werden. Diese ergeben sich aus den Definitionen des Schemas sowie der Semantik der *F-Logic*, hierbei insbesondere aus den Axiomen.

Die folgende Liste enthält alle zu berücksichtigenden Aspekte für die Transformation des Problems:

- Menge SG_D der Signaturen für Klassenattribute und für Klassen. Dabei ist bei Signaturen für Klassenattribute zu berücksichtigen:
  (1) Strikte Typung, so gilt etwa: Wenn $o[A \Rightarrow R]$, $o : c$ und $o[A \rightarrow o']$, dann $o' : R$.
  (2) *Well-typedness*-Axiome: Modelle werden auf korrekt getypte Interpretationen beschränkt (vgl. Def. 2.2).
  (3) Polymorphe Attribute: Wird ein Attribut für zwei oder mehr Klassen definiert, die bezüglich der durch den transitiven Abschluss der Klassenhierachie gegebenen Ordnungsrelation $\leq_{HR_D}$ nicht vergleichbar sind, so handelt es sich um ein polymorphes Attribut.
  (4) *Not-null*-Axiom: Attribute liefern auf Objekten, auf denen sie definiert sind, immer einen Wert zurück (vgl. Def. 2.2).
- Die Mengen CL_D, AT_D und pp_f (Klassen-, Attributs-, Objekt-ID-Terme) sind in *F-Logic* disjunkt, $CL_D \cap AT_D = \emptyset$ (vgl. Def. 2.1 und 2.4).
- Menge HR_D der Klassen-Hierarchie. Dabei ist zu achten auf
  (1) Vererbung der Signaturen von Klassen und Klassenattributen (vgl. 2.1).
  (2) Die Klassenhierarchie ist azyklisch: Hierarchien wie $\{a :: d, d :: c\}$ sind in der *F-Logic* nicht erlaubt. Dies wird durch $HR_D \models UN$ (Beachtung der *unique-names*-Axiome) sichergestellt (vgl. [BP03, S. 400]).
- Menge der semantischen Bedingungen SC_D (als Aussagen).
  (1) Klasseninklusionsabhängigkeiten (CICs) $c \subset d$:
  Aufgrund der *unique-names axioms* gilt $c \subset\neq d$ (vgl. Def. 2.8)!
  (2) Onto-Abhängigkeiten $c\{a|d\}$ setzen gewisse Beschränkungen bezüglich der beteiligten Klassen voraus: $c, d \in CL_D$, $a \in AT_D$ (vgl. Def. 2.9).

(3) Pfadabhängigkeiten für eine Klasse c: $c(p_1 \dots p_k \rightarrow p_{k+1} \dots p_n)$ erfordern für alle beteiligten Pfade $p_i \in \text{PathFuncs}_D(c)$, $i \in \{1, \dots, n\}$. Die Pfade müssen für die Ausgangsklasse c definiert sein (vgl. Def. 2.16).

- *Unique-name*-Axiome: Objekte, Klassen und Attribute, die bezüglich aller Merkmale übereinstimmen, werden miteinander identifiziert (vgl. Def. 2.2).
- Begriff der Extension $f \in \text{ext}(D)$, $f = \langle pp_f | ob_f \rangle$ (vgl. [BP03, S. 400f.]).
- Begriff der Instanz (vgl. Def. 2.6).

5.3 Modellierung der F-Logic in $\mathcal{DLR}_{reg,ifd}$

In diesem Abschnitt wird die DL-Sprache $\mathcal{DLR}_{reg,ifd}$ daraufhin untersucht, inwieweit man mit ihrer Hilfe das Problem der logischen Implikation in einem *F-Logic*-Schema (siehe Kapitel 2.1) formulieren kann. Syntax und Semantik der Sprache $\mathcal{DLR}_{reg,ifd}$ wurden bereits in Abschnitt 3.2 vorgestellt, ebenso ein Entscheidungsverfahren für die logische Implikation in $\mathcal{DLR}_{reg,ifd}$-Wissensbasen.

5.3.1 Umsetzung des Schemas

Als erster Schritt wird die Möglichkeit der Übertragung eines *F-Logic*-Schemas

$$D = \langle CL_D \cup AT_D \cup HR_D \cup SG_D | SC_D \rangle$$

in eine $\mathcal{DLR}_{reg,ifd}$-Wissensbasis

$$K_D = \langle \mathcal{C}_K, \mathcal{R}_K | \mathcal{T}_K | \mathcal{A}_K | \mathcal{S}_K \rangle.$$

betrachtet.

Zur Vereinfachung der Umsetzung wird folgendes vorausgesetzt:

Bemerkung 5.1 (Vollständigkeit von HR_D und SG_D). Aufgrund der zweiten Anmerkung zu Definition 2.3 (Seite 16) kann o.B.d.A. im Folgenden davon ausgegangen werden, dass sowohl HR_D, als auch SG_D *vollständig* sind, d.h. HR_D bildet eine Halbordnung $\leq_{HR_D}$ auf CL_D ($c \leq_{HR_D} d$ bedeutet dabei: „c ist Unterklasse von d“) und SG_D enthält sämtliche – auch ererbte – Signaturen für sämtliche Attribute aus AT_D.

5.3.1.1 ID-Terme, $\mathcal{F}_D, CL_D, AT_D$

In der *F-Logic* entstammen sämtliche ID-Terme für Konstanten der Menge $\mathcal{F}_D$. Diese Menge enthält somit insbesondere die Bezeichner aller Klassen und Attribute. Innerhalb des Schemas werden sämtliche ID-Terme für Klassen in der Menge CL_D und sämtliche ID-Terme für Attribute in der Menge AT_D zusammengefasst. Im Schema einer $\mathcal{DLR}_{reg,ifd}$-Wissensbasis entstammen sämtliche Bezeichner für Konzepte und Relationen einer Menge von Bezeichnern $\mathcal{F}_K$. Hiervon werden als Teilmengen die Mengen $\mathcal{C}_K$ und $\mathcal{R}_K$ als die Mengen der Konzept- und der Relationsbezeichner im Schema K aufgeführt.

Um die Lesbarkeit zu verbessern, erfolgt die Konstruktion von K_D unter Zuhilfenahme sogenannter *Widerparte* der ID-Terme. Der Widerpart eines ID-Terms $x[] \in \mathcal{F}_D$ ist der Bezeichner x^w:

Definition 5.1 (Widerpart eines ID-Terms). *Sei $x \in \mathcal{F}_D$ ein ID-Term. Dann ist der Widerpart von $x[]$ definiert durch $x^w \in \mathcal{F}_K$.*

Bei dieser Darstellung finden für einander entsprechende Klassen-ID-Terme und Konzeptbezeichner, Attributs-ID-Terme und Relationsbezeichner, Objekt-ID-Terme und Individuenbezeichner die selben Buchstaben Verwendung:

- Klassen-ID-Terme $a[], b[], \ldots, g[]$
  durch Widerpart-Konzeptbezeichner $a^w, b^w, \ldots g^w$
- Attributs-ID-Terme $r[], s[], t[]$
  durch Widerpart-Relationsbezeichner r^w, s^w, t^w
- Objekt-ID-Terme $o[], p[], q[]$
  durch Widerpart-Individuenbezeichner o^w, p^w, q^w

Auch für die Mengen der Klassen- und Attributs-ID-Terme wird ein Widerpart definiert:

Definition 5.2 (Widerpart von $\mathcal{F}_D$, CL_D und AT_D). *Als Widerpart der Mengen ID-Term-Konstanten, der Klassen- und Attributs-ID-Terme werden definiert:*

(1) $\mathcal{F}_D^w := \{x^w \mid x[] \in \mathcal{F}_D\}$

(2) $\mathcal{C}_D^w := \{c^w \mid c[] \in CL_D\}$

(3) $\mathcal{R}_D^w := \{r^w \mid r[] \in AT_D\}$

Diese Definitionen bedeuten intuitiv folgendes:

(1) Für jeden ID-Term der *F-Logic* enthält $\mathcal{F}_D^w$ einen entsprechenden Bezeichner.

(2) Jede Klasse $c[]$ der *F-Logic* wird durch ein gleichnamiges primitives Konzept c^w in der DL dargestellt.

(3) Jedes Attribut $a[]$ der *F-Logic* wird durch eine gleichnamige primitive Relation a^w in der DL vertreten. Dabei werden Attribute unterschiedlicher Signaturen aus der *F-Logic* durch ein und dieselbe Relation in der DL dargestellt.

Für die Konstruktion des Widerparts der übrigen Komponenten von K_D werden ggf. weitere Relations- und Konzeptbezeichner benötigt.

5.3.1.2 Klassenhierarchie HR_D

Der Klassenhierarchie HR_D eines *F-Logic*-Schemas entspricht in einem DL-Schema eine Teilmenge $HR_D^w \subseteq \mathcal{T}_K$ der TBox. Entsprechend werden als Widerpart der variablenfreien *class is-a assertions* in der Klassenhierarchie Konzeptinklusionszusicherungen zwischen primitiven Konzepten Eingang in die TBox $\mathcal{T}_K$ finden.

Definition 5.3 (Widerpart von HR_D). *Der Widerpart der Klassenhierarchie HR_D wird wie folgt definiert:*

$$HR_D^w := \{(c^w \sqsubseteq d^w) \mid (c[] :: d[]) \in HR_D\}.$$

Wiederum wird $\mathcal{T}_K$ neben dem Widerpart HR_D^w von HR_D zusätzliche Konzept- und Relationsinklusionszusicherungen erhalten, insbesondere solche, welche die Funktionalität des Widerparts der Attribute, die semantischen Bedingungen und die Bedingungen der Axiome erzwingen sollen.

5.3.1.3 Signaturen SG_D

Die Menge SG_D besteht aus variablenfreien Molekülen mit skalaren Signaturausdrücken. Diese geben die Signaturen der Klassen und Attribute in D an. Der Widerpart der Signaturen wird aus Relations- und Konzeptinklusionszusicherungen bestehen, die in die Tbox $\mathcal{T}_K$ der $\mathcal{DLR}_{reg,ifd}$-Wissensbasis K_D Aufnahme finden.

Bei der Übertragung der Signaturen in die $\mathcal{DLR}_{reg,ifd}$-Wissensbasis K_D müssen folgende Punkte beachtet werden:

- Signaturen können durch die Klassenhierarchie HR_D an Unterklassen vererbt werden. Da HR_D und SG_D nach Voraussetzung vollständig sind (Bemerkung 5.1), muss dies nicht gesondert berücksichtigt werden.
- Es können polymorphe Attribute auftreten, d.h. ein Attributs-ID-Term erscheint in der Signatur mindestens zweier Klassen, die bezüglich $\leq_{HR_D}$ unvergleichbar sind.

Definition 5.4 (Widerpart von SG_D). *Skalare Signaturausdrücke und die Menge der Signaturen SG_D werden wie folgt behandelt:*

(1) Der Widerpart einer Signatur $\sigma = (a[r \Rightarrow b]) \in SG_D$, mit $a[], b[] \in CL_D$, $r[] \in AT_D$ wird definiert durch

$$\begin{aligned} \sigma^w \quad := \{ \quad & (a^w \sqsubseteq (\leq 1\,[2]\,r^w) \sqcap (\geq 1\,[2]\,r^w)) && (5.1)\\ & (a^w \sqsubseteq (\forall r^w|_{[1][2]}.b^w)) && (5.2)\\ & ((\exists\,(r^w)^-.a^w) \sqsubseteq b^w)\}. && (5.3) \end{aligned}$$

(2) Der Widerpart von $SG_D = \{\phi_1, \ldots, \phi_n\}$ ist definiert durch

$$SG_D^w := \phi_1^w \cup \ldots \cup \phi_n^w \cup \Phi,$$

wobei

$$\Phi := \{(r^w \sqsubseteq \bigsqcup_{(c_i[r \Rightarrow a_i]) \in SG_D} [(1/2 : c_i^w) \sqcap (2/2 : a_i^w)]) \mid r \in AT_D\}. \qquad (5.4)$$

Bedeutung der Zusicherung in (1) und (2):

Zusicherung 5.1 gewährleistet die Funktionalität von r^w auf a^w. Das heißt, jedes Individuum aus a^w tritt genau einmal in der ersten Komponente von r^w auf. Damit wird auch das NN-(*Not-Null-*)Axiom erfüllt.

Zusicherung 5.2 garantiert, dass a^w ausschließlich solche Individuen enthält, die durch r^w auf Individuen aus b^w abgebildet werden und erzwingt so, dass Attributswerte vom definierten Typ sind. Diese Bedingung entspricht dem WT_2-Axiom (vgl. Abschnitt 5.3.3.2).

Zusicherung 5.3 sorgt dafür, dass b^w tatsächlich das gesamte Bild $r^w(a^w)$ enthält.

Die Menge Φ von Zusicherungen in 5.4 bildet die Signaturen für jedes Attribut $r[] \in AT_D$ ab. Ist $r[]$ ein polymorphes Attribut, so werden durch diese Übertragung unterschiedliche „Partitionen“ innerhalb des Widerparts von $r[]$ erzeugt. Jeder dieser Partitionen entspricht dann genau eine der Signaturen von $r[]$. Somit wird dem WT_1-Axiom Rechnung getragen (vgl. Abschnitt 5.3.3.2).

5.3.1.4 Semantische Bedingungen SC_D

Die semantischen Bedingungen für D liegen als Formeln $\gamma \in SC_D$ vor. Auch sie erhalten jeweils einen Widerpart in K.

Definition 5.5 (Widerpart von SC_D). *Sei $SC_D = \{\gamma_1, \ldots, \gamma_m\}$. Dann wird der Widerpart von SC_D wie folgt definiert:*

$$SC_D^w := \gamma_1^w \cup \ldots \cup \gamma_m^w,$$

wobei die γ_i^w gemäß Definitionen 5.6 und 5.7 gebildet werden.

5.3.1.4.1 Klasseninklusionsabhängigkeiten (CICs) Eine Klasseninklusionsabhängigkeit $\gamma = (c \subset d) \in SC_D$ besagt in Verbindung mit der vorgestellten Semantik der *F-Logic* nicht, dass c eine Unterklasse von d sei, sondern lediglich, dass jedes Objekt, das Instanz von c ist, auch Instanz von d ist.

Definition 5.6 (Widerpart eines CIC). *Sei $\gamma = (c \subset d) \in SC_D$. Dann wird der Widerpart von γ wie folgt definiert:*

$$\gamma^w := \{(c^w \sqsubseteq d^w)\}.$$

5.3.1.4.2 Onto-Abhängigkeiten (OCs) Eine Onto-Abhängigkeit $\gamma = c\{r|d\}$ in D garantiert die „Erreichbarkeit“ aller Instanzen der Klasse d von Instanzen der Klasse c über das Attribut r. In *Description Logics* entspricht dieser Erreichbarkeit eine Werterestriktion für eine inverse Rolle. In $\mathcal{DLR}_{reg,ifd}$ lassen sich inverse Rollen durch den Projektionskonstruktor $R|_{[i][j]}$ und die Werterestriktion für reguläre Ausdrücke $\forall E.C$ modellieren:

Definition 5.7 (Widerpart eines OC). *Sei $\gamma = c\{r|d\} \in SC_D$. Dann wird der Widerpart von γ wie folgt definiert:*

$$\gamma^w := \{(d^w \sqsubseteq \forall \underbrace{r^w|_{[2][1]}}_{=(r^w)^-}.c^w)\}.$$

5.3.1.4.3 Pfadabhängigkeiten (PFDs) Pfadabhängigkeiten lassen sich in $\mathcal{DLR}_{reg,ifd}$ nicht darstellen. Dies ergibt sich bereits aus den Untersuchungen von Calvanese et al. zu unären funktionalen Abhängigkeiten in $\mathcal{DLR}_{ifd}$ [CGL01]. Demnach führt die Erweiterung von $\mathcal{DLR}_{ifd}$ um unäre funktionale Abhängigkeiten, also Ausdrücke wie (**fd** $R\ 1 \rightarrow 2$), zur Unentscheidbarkeit von $\mathcal{DLR}_{ifd}$. Der Beweis zeigt eine polynomielle Reduktion des unentscheidbaren *unrestricted tiling problems* [vEB97] von und auf ein Konzepterfüllbarkeitsproblem in einer $\mathcal{DLR}_{ifd}$-Wissensbasis K_T. Die Konstruktion von K_T benutzt dabei folgende Menge von Konstruktoren und Zusicherungen: $(\leq\ k[i]R), \neg C, C_1 \sqcap C_2, \top_1; \neg R, R_1 \sqcap R_2, (i/n : C); C_1 \sqsubseteq C_2$, (**fd** $R\ i \rightarrow j$). Dieses Ergebnis ist direkt auf $\mathcal{DLR}_{reg,ifd}$ übertragbar.

Da Pfadabhängigkeiten echte Erweiterungen von funktionalen Abhängigkeiten darstellen, sind allgemeine (d.h. einschließlich unäre) Pfadabhängigkeiten in $\mathcal{DLR}_{reg,ifd}$ nicht formalisierbar. Allenfalls kann versucht werden, nicht-unäre Pfadabhängigkeiten, also solche mit mehr als einer Prämisse, darzustellen.

Dazu sei an dieser Stelle ein Ansatz zur Erzeugung eines Widerparts für eine nicht-unäre PFD $\gamma = c(\mathrm{Pf}_1 \cdots \mathrm{Pf}_n \rightarrow \mathrm{Pf}_{n+1}) \in SC_D$, für $n > 1$ dargestellt:

Zunächst kann man sich auf die oben angegebene Form von PFDs mit nur einer Konsequenz beschränken, da jede PFD mit $k > 1$ Konzequenzen durch k PFDs mit denselben Prämissen und einer einzigen Konzequenz ersetzt werden kann (vgl. Bemerkung 3.3).

Für die Pfadfunktionen in γ gelte:

$$\mathrm{Pf}_i = r_i^1.r_i^2.\ \ldots\ .r_i^{l_i}.\mathrm{Id} \qquad \text{mit} \qquad l_i := \mathrm{length}(\mathrm{Pf}_i).$$

Zuerst wird eine neue, $(n+2)$-äre Relation R_γ eingeführt. Jedes Tupel dieser Relation soll später unter einer Interpretation $\mathcal{I}$ in der ersten Komponente eine Instanz $o \in c^\mathcal{I}$ und in der $(i+1)$-ten Komponente den Wert $\mathrm{Pf}_i^\mathcal{I}(o)$ enthalten.

Mittels der konzeptgebundenen Relationskonstruktion und einer Relationsinklusionszusicherung wird garantiert, dass jedes Individuum $o^w \in (c^w)^\mathcal{I}$ in der ersten Komponente eines Tupels von $R_\gamma^\mathcal{I}$ auftritt (Zusicherung 5.6).

Eine Anzahlbeschränkung für c^w über R_γ garantiert, dass innerhalb der neuen Relation die erste Komponente einen Schlüssel darstellt, also jedes Individuum $o \in (c^w)^\mathcal{I}$ höchstens einmal in der ersten Komponente von $R_\gamma^\mathcal{I}$ auftritt (Zusicherung 5.7).

Durch Einführen einer FD-Zusicherung für R_γ wird sichergestellt, dass die Kombination der Komponenten $2, \ldots, (k+1)$ die letzte Komponente $(k+2)$ funktional bestimmt (Zusicherung 5.8).

Die Pfadfunktionen Pf_i könnten als reguläre Ausdrücke durch Komposition der beteiligten Relationen $(r_i^1)^w, \ldots, (r_i^{l_i})^w$ ausgedrückt werden.

Leider ist dies in $\mathcal{DLR}_{reg,ifd}$ *nicht* möglich, da zwar reguläre Ausdrücke, nicht jedoch Relationen durch den Kompositionsoperator „$\circ$“ verknüpft werden können. Der Widerpart jeder Pfadfunktionen Pf_i wird also durch Komposition der Projektionen der Relationen $(r_i^1)^w, \ldots, (r_i^{l_i})^w$ als regulärer Ausdruck dargestellt (rechte Seite von Zusicherung 5.8). Für den Fall $\mathrm{Pf}_i = .\mathrm{Id}$ ergibt sich als Widerpart $\underbrace{(1/2 : c^w)|_{[1][1]}}_{=\mathrm{id}(c^w)}$.

Es fehlt noch die Identifizierung der Widerparte jeder Pfadfunktion Pf_i mit der Projektion der Relation R_γ auf ihre erste und $(i+1)$-te Komponente. Die Projektion von R_γ erzeugt einen regulären Ausdruck (linke Seite von Zusicherung 5.8).

Zusicherung 5.8 zeigt eine Inklusionszusicherung, die der angesprochenen Identifizierung entspricht ($E_1 \equiv E_2$ steht hier für $E_1 \sqsubseteq E_2, E_2 \sqsubseteq E_1$).

Leider sind in $\mathcal{DLR}_{reg,ifd}$ Inklusionszusicherungen zwischen regulären Ausdrücken nicht möglich. Damit scheitert auch die Darstellung nicht-unärer PFDs.

$$R_\gamma \sqsubseteq (1/(n+2) :\ c^w) \tag{5.5}$$

$$c^w \sqsubseteq (\leq 1\,[1]\,R_\gamma) \tag{5.6}$$

$$R_\gamma|_{[1][i+1]} \equiv \underbrace{(1/2 : c^w)|_{[1][1]}}_{=\mathrm{id}(c^w)} \circ \underbrace{(r_i^1)^w|_{[1][2]} \circ \ldots \circ (r_i^{l_i})^w|_{[1][2]}}_{=\mathrm{Pf}_i{}^w} \tag{5.7}$$

$$(\mathrm{fd}\ R_\gamma[2]\ldots[n+1] \to [n+2]) \tag{5.8}$$

für $1 \leq i \leq n+2$.

In Abschnitt 5.7 dieser Arbeit wird ein Versuch zur Erweiterung von $\mathcal{DLR}_{reg,ifd}$ um Inklusionszusicherungen für reguläre Ausdrücke dargestellt. Mit dieser Erweiterung wäre die Darstellung nicht-unärer Pfadabhängigkeiten in einer $\mathcal{DLR}_{reg,ifd}$-Wissensbasis möglich.

5.3.1.5 Widerpart eines Schemas

Nachdem die Bildung der Widerparte der einzelnen Bestandteile eines *F-Logic*-Schemas in $\mathcal{DLR}_{reg,ifd}$ vorgestellt wurde, kann nun die Definition des Widerparts des gesamten Schemas vorgenommen werden:

Definition 5.8 (Widerpart eines *F-Logic*-Schemas). *Der Widerpart eines* F-Logic-*Schemas* $D = \langle CL_D \cup AT_D \cup HR_D \cup SG_D | SC_D \rangle$ *mit* $PFD_D = \emptyset$ *ist definiert durch*

$$K_D = \langle CL_D^w, AT_D^w | HR_D^w \cup SG_D^w \cup SC_D^w | \emptyset | \emptyset \rangle,$$

Bezüglich der Komplexität der Bildung des Widerparts eines *F-Logic*-Schemas kann man feststellen:

Satz 5.1 (Komplexität der Bildung des Widerparts).
Sei $D = \langle CL_D \cup AT_D \cup HR_D \cup SG_D | SC_D \rangle$ *ein beliebiges* F-Logic-*Schema mit* $PFD_D = \emptyset$. *Dann ist die Konstruktion des* $\mathcal{DLR}_{reg,ifd}$*-Widerparts von* D,

$$K_D = \langle CL_D^w, AT_D^w | HR_D^w \cup SG_D^w \cup SC_D^w | \emptyset | \emptyset \rangle.$$

mit in der Größe $|D|$ *von* D *polynomiellem Zeitaufwand möglich.*

Beweis: Zuerst wird aus dem Schema D ein vollständiges Schema D' gebildet, d.h. ein Schema mit vollständiger Klassenhierarchie $HR_{D'}$ und vollständiger Signaturenmenge $SG_{D'}$. Dies ist mit in $|D|$ polynomiellem Zeitaufwand möglich, indem zuerst der transitive Abschluss $HR_{D'}$ der *class is-a*-Relationen in HR_D gebildet (Aufwand: kubisch in $|HR_D|$) und anschließend unter Verwendung von $HR_{D'}$ die Signaturenmenge SG_D zu einer vollständigen Signaturenmenge $SG_{D'}$ erweitert wird (Aufwand: polynomiell in $|HR_{D'}| \cdot |SG_D|$).

Dann ist die Bildung der Widerparte von $AT_{D'}$, $CL_{D'}$, $HR_{D'}$, $SG_{D'}$ und $SC_{D'}$ mit in $|D'|$ linearem Zeitaufwand möglich. Die Konstruktion von K_D bedarf also insgesamt eines in $|D|$ polynomiellen Zeitaufwands. ∎

5.3.2 Widerpart einer Extension

Zu einem *F-Logic*-Schema D ohne Pfadabhängigkeiten kann man wie gezeigt als Widerpart eine $\mathcal{DLR}_{reg,ifd}$-Wissensbasis K_D konstruieren. Zu einer Extension von D kann man wiederum als $\mathcal{DLR}_{reg,ifd}$-Widerpart eine Interpretation konstruieren:

Definition 5.9 (Widerpart einer Extension). *Für eine* F-Logic-*Extension*

$$f = \langle pp_f | ob_f \rangle \in ext(D)$$

zum Schema D gemäß Definition 2.4, in dem keine Pfadabhängigkeit vereinbart ist, wird als ein Widerpart eine Interpretation $f^w = \mathcal{I}_f = (\Delta^{\mathcal{I}_f}, .^{\mathcal{I}_f})$ zum Schema K_D konstruiert, so dass gilt:

(1) Jeder Objekt-ID-Term wird durch einen Widerpart in der Interpretationsdomäne vertreten:

$$o^w \in \Delta^{\mathcal{I}_f} \qquad gdw. \qquad \exists c[] \in CL_D : \ (o : c) \in pp_f.$$

(2) Während in F-Logic *die Vererbung der Klassenzugehörigkeiten automatisch erfolgt, müssen in $\mathcal{DLR}_{reg,ifd}$ alle Objekte ausdrücklich in die Konzepte aller Oberklassen eingetragen werden:*[1]

$$o^w \in (c'^w)^{\mathcal{I}_f} \qquad gdw. \qquad \exists c' \in CL_D : (c \leq_{HR_D} c') \wedge o : c \in pp_f.$$

Da in F-Logic *jede Zugehörigkeit eines Objekts o zu einer Klasse c entweder direkt durch eine explizite* object is-a assertion *$o : c$ in pp_f oder aber indirekt durch eine* object is-a assertion *$o : c_1$ in pp_f in Verbindung mit einer Kette von* class is-a assertions *$\{c_1 :: c_2, c_2 :: c_3, \ldots, c_n :: c\} \subseteq HR_D$ ausgedrückt wird, werden so alle Klassenzugehörigkeiten von o zuverlässig auf Zugehörigkeiten zu primitiven Konzepten abgebildet.*

(3) Die skalaren Datenausdrücke in der Attribution ob_f werden folgendermaßen bedacht:

$$(o^w, v^w) \in (a^w)^{\mathcal{I}_f} \qquad gdw. \qquad (o[a \rightarrow v]) \in ob_f.$$

Umgekehrt kann man zu einer gegebenen Interpretation für einen D-Widerpart K_D folgendermaßen eine *simulierte Extension* von D konstruieren:

Satz 5.2 (Simulierte Extension). *Zu einer Interpretation $\mathcal{I}$ für den Widerpart K_D eines* F-Logic-*Schemas D wird eine* simulierte Extension

$$f_{\mathcal{I}} = \langle pp_{f_{\mathcal{I}}} | ob_{f_{\mathcal{I}}} \rangle \in ext(D)$$

wie folgt gebildet:

[1] Nach Voraussetzung der Vollständigkeit von HR_D und SG_D im *F-Logic*-Schema D bildet HR_D bereits eine Halbordnung $\leq_{HR_D}$ auf CL_D.

(1) Die Population $pp_{f_\mathcal{I}}$ wird wie folgt gebildet: $(o:c) \in pp_{f_\mathcal{I}}$ gdw. $c^w \in CL_K^w$ und $o^w \in (c^w)^\mathcal{I}$.

(2) Die Attribution $ob_{f_\mathcal{I}}$ wird wie folgt festgelegt: Genau dann, wenn ein Tupel $(o^w, (o')^w) \in (a^w)^\mathcal{I}$ für ein Attribut $a^w \in AT_K^w$ existiert, gilt $(o[a \rightarrow o']) \in ob_{f_\mathcal{I}}$.

5.3.3 Umsetzung der Axiome

Die Formelmenge AX der Axiome stellt sicher, dass jede Instanz des *F-Logic*-Schemas D den darin spezifizierten Bedingungen für Klassen und Attribute genügt.

Die Axiome können im $\mathcal{DLR}_{reg,ifd}$-Widerpart des Schemas folgendermaßen berücksichtigt werden:

5.3.3.1 Not-Null-Axiom (NN)

Das *Not-Null-Axiom* $NN = \{\forall C \forall A \forall O \exists V O[A \rightarrow V] \longleftarrow C[A \Rightarrow ()] \wedge O:C\}$ garantiert in der *F-Logic*, dass ein für eine Klasse deklariertes Attribut auf Instanzen dieser Klasse immer einen Wert zurückliefert.

Die Einhaltung dieser Anforderung wird in K_D jedoch bereits durch die Paare von Anzahlbeschränkungen in den CICs

$$a^w \sqsubseteq (\leq 1\,[2]\,r^w) \sqcap (\geq 1\,[2]\,r^w)$$

im Widerpart von SG_D (Definition 5.4) in Verbindung mit der Annahme der Vollständigkeit von HR_D und SG_D sichergestellt.

5.3.3.2 Well-Typedness-Axioms (WT)

Die *Well-Typedness-Axioms* bestehen aus zwei Axiomen:

$$WT = WT_1 \cup WT_2 :$$

$$\begin{aligned} WT_1 &= \{\forall O \forall A \forall V \exists C: \quad (C[A \Rightarrow ()] \wedge O{:}C) \longleftarrow O[A \rightarrow V]\} \\ WT_2 &= \{\forall O \forall A \forall V \forall C \forall R: \quad V{:}R \longleftarrow C[A \Rightarrow R] \wedge O{:}C[A \rightarrow V]\}. \end{aligned}$$

WT_1 erzwingt, dass wenn ein Attribut auf einem Objekt definiert ist und einen Wert liefert, das Objekt zu einer Klasse gehört, für welche das Attribut deklariert wurde.

Bezeichne $\pi_{[i]}(R)$ die Projektion einer Relation R auf ihre i-te Komponente. Damit ergibt sich folgender Ansatz:

$$\forall r^w \forall v^w: \quad (o^w, v^w) \in (r^w)^\mathcal{I} \Rightarrow \exists c^w:\ o^w \in (c^w)^\mathcal{I} \wedge (c^w)^\mathcal{I} \subseteq \pi_{[1]}((r^w)^\mathcal{I}).$$

In K_D wird WT_1 durch die Übersetzung der skalaren Signaturausdrücke SG_D^w, der Klassenhierarchie HR_D^w und der Definition des Widerparts der Extension (siehe Abschnitt 5.3.2) erzwungen.

Ist WT_1 in der Extension f für ein Attribut r nicht erfüllt, so kann der Widerpart zu f kein Modell von K_D sein, da dann die Konzeptinklusionszusicherung

$$(r \sqsubseteq \bigsqcup_{(c_i[r \Rightarrow a_i]) \in SG_D} [(1/2 : c_i^w) \sqcap (2/2 : a_i^w)])$$

nicht erfüllbar ist.

WT_2 erzwingt, dass Werte, die ein Attribut liefert, einem vereinbarten Typ entsprechen. Dies wird jedoch bereits durch die Übersetzung der skalaren Signaturausdrücke und der Klassenhierarchie sichergestellt, und zwar gerade durch die CICs

$$\exists\, (r^w)^-.a^w \sqsubseteq b^w.$$

Es fällt kein zusätzlicher Aufwand für die Überprüfung der WT-Axiome an.

5.3.3.3 Unique-Name-Axioms (UN)

Während die *Unique-Name*-Bedingung in vielen *Description Logic*-Sprachen vorausgesetzt wird (sogenannte *unique-name assumption*), ist dies bei $\mathcal{DLR}_{reg,ifd}$ nicht der Fall. Da die UN-Axiome bei der Konstruktion von K_D noch nicht berücksichtigt wurden, muss die Einhaltung der UN-Axiome durch ein Modell von K_D explizit überprüft werden.

In *F-Logic* wurden die UN-Axiome folgendermaßen formalisiert:

$$UN = \{\overset{\circ}{\neq}(t, t') | t, t' \in U(\mathcal{F}), t \not\equiv t'\}.$$

UN kann man wird in drei disjunkte Mengen teilen: Nämlich die Formeln bezüglich der UN-Aussage für alle Klassen-ID-Terme (1), für alle Attributs-ID-Terme (2) und für alle Objekt-ID-Terme (3).

(1) Sei K_D erfüllbar. Bei Verstoß gegen eine UN-Annahme für Klassen ist $K_D \cup \mathcal{T}_{UN_1}$ für mindestens eine der Erweiterungen um ein nachfolgend definiertes $\mathcal{T}_{UN_1}$ erfüllbar:

$\mathcal{T}_{UN_1} = \{(c_1^w \sqsubseteq c_2^w), (c_2^w \sqsubseteq c_1^w)\}$; $c_1^w, c_2^w \in CL_D^w, c_1^w \not\equiv c_2^w$.

Es sind $|CL_D|^2$ mögliche Kombinationen zu prüfen. Nur wenn $K_D \cup \mathcal{T}_{UN_1}$ für alle Kombinationen c_1^w, c_2^w unerfüllbar ist, wird nicht gegen das UN-Axiom für Klassen verstoßen.

(2) Bei Verstoß gegen eine UN-Annahme für Attribute ist mindestens eine der Erweiterungen von K_D um ein nachfolgend definiertes $\mathcal{T}_{UN_2}$ unerfüllbar:
$\mathcal{T}_{UN_2} = \{(r_1^w \sqsubseteq r_2^w), (r_2^w \sqsubseteq r_1^w)\}$; $r_1^w, r_2^w \in AT_D^w, r_1^w \not\equiv r_2^w$.
(Es sind $|AT_D|^2$ mögliche Kombinationen zu prüfen.)

(3) Zum Überprüfen auf Verstöße gegen die UN-Annahme für Objekte: Konstruiere als Gegenbeispiel eine ABox $\mathcal{A}_K$ mit zwei in K_D neuen Skolemkonstanten x, y und mit $(x \neq y) \in \mathcal{A}_K$. Die ABox stellt durch weitere ABox-Zusicherungen sicher, dass zwei Objekte in einer Interpretation, die K_D erfüllt, gleiche Eigenschaften haben und somit gegen die UN-Annahme verstoßen:

- Für alle $c^w \in CL_D^w$:
  Entweder $\{c^w(x), c^w(y)\} \subseteq \mathcal{A}_K$ oder $\{\neg c^w(x), \neg c^w(y)\} \subseteq \mathcal{A}_K$.
  (Objekte x, y haben identischen Klassenzugehörigkeiten)
- Für $N = |\Delta^{\mathcal{I}}|$, $z_k, 1 \leq k \leq N$, sind neue Skolemkonstanten in K_D:
  $\{(z_i \neq z_j) | 1 \leq i < j \leq N\} \subseteq \mathcal{A}_K$. Ist f bekannt, so auch
  $N = |\Delta^{\mathcal{I}}| = |pp_f|$.
- Für alle $a^w \in \mathcal{AT}_K, 1 \leq i \leq N$:
  Entweder $\{a^w(x, z_i), a^w(y, z_i)\} \subseteq \mathcal{A}_K$ oder $\{\neg a^w(x, z_i), \neg a^w(y, z_i)\} \subseteq \mathcal{A}_K$.
  (Objekte x, y haben identische Bildbereiche bzgl. aller Attribute)

  Entweder $\{a^w(z_i, x), a^w(z_i, y)\} \subseteq \mathcal{A}_K$ oder $\{\neg a^w(z_i, x), \neg a^w(z_i, y)\} \subseteq \mathcal{A}_K$.
  (Objekte x, y haben identische Urbildbereiche bzgl. aller Attribute)

Dann muss K_D mit jeder so definierten ABox unerfüllbar sein, damit es keine zwei bezüglich aller Eigenschaften gleichen Objekte geben kann.

Das Testen polynomiell vieler Erweiterungen (nämlich $|\mathcal{CO}_{\mathcal{K}}|^2 + |\mathcal{AT}_{\mathcal{K}}|^2$) genügt dann zur Überprüfung auf Verstöße gegen die UN-Annahme für Klassen und Attribute. Zwar gibt es für (3) exponentiell viele, nämlich $|AT_K| \cdot |CO_K| \cdot 2^{|\Delta|+1}$, Wahlmöglichkeiten für die ABox, aber alle Erweiterungen sind bezüglich ihrer Größe polynomiell in der Größe von K_D und f: Die Kosten für die Konstruktion jeder ABox betragen

$$O(\underbrace{|CL_D|}_{\text{Punkt 1}} + \underbrace{\frac{|pp_f|^2}{2}}_{\text{Punkt 2}} + \underbrace{|pp_f| \cdot |AT_D|}_{\text{Punkt 3}}).$$

Jeder einzelne Test des erweiterten Schemas K_D auf Unerfüllbarkeit ist exponentiell in der Größe der Eingabe [CGL01]. Somit benötigt die Überprüfung der UN-Axiome insgesamt einen zeitlichen Aufwand von

$$\begin{aligned} & O(|CL_D|^2 \cdot T) \\ + \quad & O(|AT_D|^2 \cdot T) \\ + \quad & O(|AT_D| \cdot |CL_D| \cdot 2^{|pp_f|+1} \cdot (|CL_D| + \tfrac{|pp_f|^2}{2} + |AT_D| \cdot |pp_f| + T)), \end{aligned}$$

wobei T der Aufwand für einen einzelnen Unerfüllbarkeitstest ist.

Für endliche Domänen ist bezüglich der Eingabe $\langle K_D, f\rangle$ also in doppelt-exponentieller Zeit überprüfbar, ob zwei Objekte in allen Eigenschaften übereinstimmen und somit ein UN-Axiom verletzen. Für unendliche Domänen ist dieses Axiom nicht überprüfbar. Seine Beachtung muss daher vorausgesetzt werden (*unique name assumption*).

Bemerkung 5.2 (*Unique name assumption*). Da es nicht möglich ist, die Einhaltung der UN-Axiome mittels des vorgestellten Tests auch für unendliche Instanzen/Modelle zu gewährleisten, wird im weiteren Verlauf dieses Abschnitts angenommen, dass alle Bezeichner/ID-Terme unterschiedliche Extensionen haben (*unique name assumption*), insbesondere also die UN-Axiome beachtet werden.

5.3.4 Vollständigkeit und Korrektheit der Reduktion

Um zu zeigen, dass die vorgestellte Modellierung der *F-Logic* (ohne Berücksichtigung von PFDs) eine polynomielle Reduktion der logischen Implikation in der *F-Logic* auf die logische Implikation in $\mathcal{DLR}_{reg,ifd}$ darstellt, müssen die Vollständigkeit (jedes erfüllbare *F-Logic*-Schema hat einen erfüllbaren $\mathcal{DLR}_{reg,ifd}$-Widerpart) und die Korrektheit (wenn der $\mathcal{DLR}_{reg,ifd}$-Widerpart eines *F-Logic*-Schemas erfüllbar ist, so ist auch das *F-Logic*-Schema erfüllbar) der Reduktion gezeigt werden.

Satz 5.3 (Vollständigkeit der Reduktion). *Sei D ein* F-Logic-*Schema nach Def. 2.1, jedoch ohne Pfadabhängigkeiten in SC_D, und K_D eine Wissensbasis als Widerpart von D wie oben konstruiert. Dann gilt:*

Ist f eine Instanz von D, so gibt es ein Modell $\mathcal{I}_f$ von K_D.

Beweis: Sei f eine Instanz des Schemas $D = \langle CL_D \cup AT_D \cup HR_D SG_D | SC_D\rangle$. Dann ist der Abschluss von f unter D ein Herbrand-Modell von $AX \cup SC_D$. Sei nun

$$K_D = \langle CL_D^w, AT_D^w | \underbrace{HR_D^w \cup SG_D^w \cup SC_D^w}_{=\mathcal{T}_K} | \underbrace{\emptyset}_{\mathcal{A}_K} | \underbrace{\emptyset}_{\mathcal{S}_K}\rangle$$

der Widerpart zu D und $\mathcal{I}_f$ der Widerpart zu f. Angenommen $\mathcal{I}$ ist kein Modell von K_D. Dann gibt es eine Zusicherung $\tau \in \mathcal{T}_K \cup \mathcal{A}_K \cup S_K$, die von $\mathcal{I}_f$ nicht erfüllt wird. Da K_D eine leere ABox und eine leere Menge semantischer Bedingungen aufweist, muss gelten: $\tau \in \mathcal{T}_K$.

Man kann unterscheiden:

(1) $\tau \in HD_D^w$
HR_D^w enthält ausschließlich Zusicherungen der Form $c^w \sqsubseteq d^w$. Dabei besteht HR_D^w nur aus solchen Zusicherungen, dass gilt: $c :: d \in HR_D$.
Nun wird τ verletzt, d.h. es gibt ein Individuum $o^w \in \Delta^{\mathcal{I}_f}$, so dass $o^w \in (c^w)^{\mathcal{I}_f}$

und $o^w \notin (d^w)^{\mathcal{I}_f}$. Nach der Konstruktion von $\mathcal{I}_f$ gilt dann $o : c \in pp_f$ und für alle Oberklassen $d'[] \in CL_D, d \leq_{HR_D} d'$ ist $o : d' \notin pp_f$, insbesondere also $o : d \notin pp_f$. Dadurch ist die Klassenhierarchie bezüglich $c :: d \in HR_D$ verletzt und f kann keine Instanz von D sein. Ein Widerspruch.

(2) $\tau \in SG_D^w$
SG_D^w besteht aus vier Arten von Zusicherungen:

(a) $(a^w \sqsubseteq (\leq 1\,[2]\,r^w) \sqcap (\geq 1\,[2]\,r^w))$ für $a[r \Rightarrow b] \in SD_D$
Sei τ oben stehende Zusicherung. Dann gibt es ein $o^w \in (a^w)^{\mathcal{I}_f}$ mit $o^w \notin (\leq 1\,[2]\,r^w)^{\mathcal{I}_f}$ oder $o^w \notin (\geq 1\,[2]\,r^w)$.
Im ersten Fall gäbe es also mindestens zwei Individuen $p^w, q^w \in \Delta^{\mathcal{I}_f}$ mit $\{(o^w, p^w), (o^w, q^w)\} \subseteq (r^w)^{\mathcal{I}_f}$. Gemäß der Konstruktion von $\mathcal{I}_f$ ist damit $\{o[r \to p], o[r \to q]\} \subseteq ob_f$. Dies steht im Widerspruch dazu, dass Attribute in der *F-Logic* einwertig sind.
Im zweiten Fall gibt es kein Individuum $p^w \in \Delta^{\mathcal{I}_f}$ mit $(o^w, p^w) \in (r^w)^{\mathcal{I}_f}$. Gemäß der Konstruktion von $\mathcal{I}_f$ gilt dann: $\nexists p[] \in \mathcal{F}_D : o[r \to p] \in ob_f$. Da $a[r \Rightarrow b] \in HR_D$ und für f das NN-Axiom gilt, ist dies widersprüchlich.

(b) $(a^w \sqsubseteq (\forall r^w|_{[1][2]}.b^w))$ für $a[r \Rightarrow b] \in SG_D$
Sei τ diese Zusicherung. Dann gibt es ein $o^w \in (a^w)^{\mathcal{I}_f}$ mit $o^w \notin (\forall r^w|_{[1][2]}.b^w)^{\mathcal{I}_f}$. Das heißt, es gibt ein $p^w \in \Delta^{\mathcal{I}_f}$, so dass $p^w \notin (b^w)^{\mathcal{I}_f}$ und $(o^w, p^w) \in (r^w)^{\mathcal{I}_f}$. Nach Konstruktion von $\mathcal{I}_f$ gibt es also $o[r \to p]$ und für alle Unterklassen $c, c \leq_{HR_D} \mathrm{b}$ gilt $p : c \notin pp_f$. Somit gilt insbesondere $o : b \notin pp_f$. Da $a[r \Rightarrow b] \in SD_D$ ist dies ein Verstoß gegen die WT-Axiome und f ist keine Instanz von D, ein Widerspruch.

(c) $((\exists\,(r^w)^-.a^w) \sqsubseteq b^w)$ für $a[r \Rightarrow b] \in SG_D$
Sei τ diese Zusicherung. Dann gibt es ein $o^w \in (\exists\,(r^w)^-.a^w)^{\mathcal{I}_f}$ mit $o^w \notin (b^w)^{\mathcal{I}_f}$. Das bedeutet, es gibt ein $p^w \in (a^w)^{\mathcal{I}_f}$ mit $(p^w, o^w) \in (r^w)^{\mathcal{I}_f}$. Aus der Konstruktion von $\mathcal{I}_f$ folgt einerseits $p[r \to o] \in ob_f$, $p : a \in pp_f$ und für alle $d \in CL_D, c \leq_{HR_D} b$ gilt $o : c \notin pp_f$ (also insbesondere $o : b \notin pp_f$). Da $a[r \Rightarrow b] \in SG_D$ und da f die WT-Axiome respektiert, gilt andererseits $o : b$, was zum Widerspruch führt.

(d) $(r^w \sqsubseteq \bigsqcup_{(c_i[r \Rightarrow a_i]) \in SG_D}[(1/2 : c_i^w) \sqcap (2/2 : a_i^w)])$
Sei τ diese Zusicherung. Dann gibt es ein $(o^w, p^w) \in (r^w)^{\mathcal{I}_f}$ mit

$$(o^w, p^w) \notin \left(\bigsqcup_{(c_i[r \Rightarrow a_i]) \in SG_D} [(1/2 : c_i^w) \sqcap (2/2 : a_i^w)] \right)^{\mathcal{I}_f}.$$

Insbesondere gibt es also kein c_i^w, a_i^w mit $(c_i[r \Rightarrow a_i]) \in SG_D$, so dass gilt $(o^w, p^w) \in [(1/2 : c_i^w) \sqcap (2/2 : a_i^w)]^{\mathcal{I}_f}$.
Aufgrund der Konstruktion von $\mathcal{I}_f$ gilt $o[r \to p] \in ob_f$. Da f den WT-Axiomen genügt, gibt es daher einen Klassen-ID-Term $e[] \in CL_D$ mit $D \cup f \models o : e$ und (da SG_D vollständig ist) eine Signatur $e[r \to g] \in SG_D$

für einen Klassen-ID-Term $g[] \in CL_D$. Nach der Konstruktion von $\mathcal{I}_f$ gilt dann: $o^w \in e^w, p^w \in g^w$, also $(o^w, p^w) \in [(1/2 : e^w) \sqcap (2/2 : g^w)]^{\mathcal{I}_f}$. Dies erzeugt einen Widerspruch.

(3) $\tau \in SC_D^w$
SC_D^w besteht aus zwei Arten von Zusicherungen

(a) $(c^w \sqsubseteq d^w)$ für CICs $(c \subset d) \in SC_D$
Sei τ diese Zusicherung. Dann gibt es $o^w \in \Delta^{\mathcal{I}_f}$, so dass $o^w \in (c^w)^{\mathcal{I}_f}$ und $o^w \notin (d^w)^{\mathcal{I}_f}$. Nach der Konstruktion von $\mathcal{I}_f$ gilt dann $o : c \in pp_f$ und $o : d \notin pp_f$. Da f Instanz von D ist und $(c \subset d) \in SC_D$, gilt insbesondere $\forall O : \; O : d \longleftarrow O : c$. Ein Widerspruch.

(b) $(d^w \sqsubseteq \forall\, r^w|_{[2][1]}.c^w)$ für OCs $c\{r|d\} \in SC_D$
Sei τ diese Zusicherung. Dann gibt es ein $o^w \in (d^w)^{\mathcal{I}_f}$ mit $o^w \notin (\forall\, r^w|_{[2][1]}.c^w))^{\mathcal{I}_f}$. Das heißt, es gibt ein $p^w \in \Delta^{\mathcal{I}_f}$, so dass $(p^w, o^w) \in (r^w)^{\mathcal{I}_f}$, aber $p^w \notin (c^w)^{\mathcal{I}_f}$. Dann gilt aufgrund der Konstruktion von $\mathcal{I}_f$, dass $o : d \in pp_f$, $p[r \to o] \in ob_f$ und für alle Klassen-ID-Terme $e[] \in CL_D, c \leq_{HR_D} e$ gilt $p : e \notin pp_f$, also insbesondere $p : c \notin pp_f$. Nun ist f Instanz von D und die semantischen Bedingungen werden nicht verletzt. Daher gilt $\forall V \exists O : \; (O : c[r \to V] \wedge c[r \Rightarrow ()]) \longleftarrow V : d$, insbesondere also $(p : c[r \to o] \wedge c[r \Rightarrow ()]) \longleftarrow o : d$, also $p : c$, was zum Widerspruch führt.

Somit ist $\mathcal{I}_f$ ein Modell von D, womit die Behauptung folgt. ∎

Satz 5.4 (Korrektheit der Reduktion). *Sei D ein* F-Logic-*Schema nach Def. 2.1, jedoch ohne Pfadabhängigkeiten in SC_D, und K_D eine Wissensbasis als Widerpart von D wie oben konstruiert. Dann gilt unter der* unique name assumption*:*

Gibt es ein Modell $\mathcal{I}$ von K_D, so ist D erfüllbar.

Beweis: (Methode: Konstruiere aus $\mathcal{I}$ eine Extension $f_{\mathcal{I}}$ von D und zeige, dass $f_{\mathcal{I}}$ Instanz von D ist.)

Konstruiere gemäß Definition 5.2 die simulierte Extension $f_{\mathcal{I}}$ zum Modell $\mathcal{I}$ von K_D.

Angenommen, $f_{\mathcal{I}}$ sei keine Instanz von D, d.h. $f_{\mathcal{I}} \notin \mathrm{sat}(D)$, d.h. $\mathrm{compl}_D(f_{\mathcal{I}})$ sei kein H-Modell der Axiome AX und der semantischen Bedingungen SC_D.

Dann liegt o.B.d.A. einer der folgenden Fälle vor:

(1) HR_D wird verletzt, $\exists C_1 \exists C_2 \exists O : \quad (C_1 :: C_2) \wedge (O : C_1) \wedge \neg(O : C_2)$, d.h. es gibt in CL_D mindestens zwei Klassen c_1 und c_2, mit $(c_1 :: c_2) \in HR_D$, so dass es ein Objekt o gibt, das zwar Instanz der Klasse c_1 $((o : c_1) \in ob_{f_{\mathcal{I}}})$, jedoch nicht Instanz der Klasse c_2 $((o : c_2) \notin ob_{f_{\mathcal{I}}})$ ist.

Aufgrund der Konstruktion von $f_{\mathcal{I}}$ muss dann gelten: $o^w \in (c_1^w)^{\mathcal{I}}$ und $o^w \notin (c_2^w)^{\mathcal{I}}$. Aus $(c_1 :: c_2) \in HR_D$ folgt aufgrund der Konstruktion von

K_D, dass $(c_1^w \sqsubseteq c_2^w) \in \mathcal{T}_K$. Da nach Annahme $\mathcal{I}$ Modell von K_D ist, folgt $(c_1^w)^{\mathcal{I}} \subseteq (c_2^w)^{\mathcal{I}}$. Ein Widerspruch.

(2) SG_D wird verletzt, d.h. es wird gegen das NN- oder das WT-Axiom verstoßen (s.u.).

(3) SC_D wird verletzt. Zwei Fälle:

(a) CIC_D, d.h. $\exists C_1 \exists C_2 \exists O: \quad (C_1 \subset C_2) \wedge (O{:}C_1) \wedge (\neg O{:}C_2)$.
Es gelte nunmehr $(c_1 \subset c_2) \wedge (o{:}c_1) \wedge \neg o{:}c_2$. Aufgrund der Konstruktion von $f_{\mathcal{I}}$ gilt also $o^w \in (c_1^w)^{\mathcal{I}}$ und $o^w \notin (c_2^w)^{\mathcal{I}}$. Aus $(c_1 \subset c_2) \in CIC_D$ ergibt sich für K_D aufgrund der Konstruktion: $(c_1^w \sqsubseteq c_2^w) \in \mathcal{T}_K$. Da $\mathcal{I}$ Modell von K_D ist, gilt $(c_1^w)^{\mathcal{I}} \subseteq (c_2^w)^{\mathcal{I}}$, also folgt insbesondere aus $o^w \in (c_1^w)^{\mathcal{I}}$ auch $o^w \in (c_2^w)^{\mathcal{I}}$. Ein Widerspruch.

(b) OC_D, d.h. $\exists C \exists D \exists A \exists V \forall O: \quad (V : D) \wedge \neg(O : C[A \rightarrow V] \wedge C[A \Rightarrow ()])$.
Es sei das OC $a\{r|b\}$ verletzt und es gelte $o : b \in pp_{\mathcal{I}}$. Dann gibt es zwei mögliche Fälle:

i. Es gibt kein $o' : a \in pp_{f_{\mathcal{I}}}$ für das gilt $o'[r \rightarrow o] \in ob_{f_{\mathcal{I}}}$. Aufgrund der Konstruktion von $f_{\mathcal{I}}$ gilt dann $o^w \in (b^w)^{\mathcal{I}}$ und es gibt kein $p^w \in (a^w)^{\mathcal{I}}$ mit $(p^w, o^w) \in (r^w)^{\mathcal{I}}$. Aufgrund der Konstruktion von K_D gilt:
$(b^w \sqsubseteq \forall r^w|_{[2][1]}.a^w) \in \mathcal{T}_K$. Da $\mathcal{I}$ Modell von K_D ist und $p^w \ni (b^w)^{\mathcal{I}} \neq \emptyset$ gilt auch $\exists q^w \in (a^w)^{\mathcal{I}} : (o^w, q^w) \in (r^w|_{[2][1]})^{\mathcal{I}}$ und damit $(q^w, o^w) \in (r^w)^{\mathcal{I}}$, also $q : a \in pp_{\mathcal{I}}$ und $q[r \rightarrow o] \in ob_{\mathcal{I}}$, ein Widerspruch.

ii. Es gibt keinen Klassen-ID-Term $a[] \in AT_D$ mit $a[r \Rightarrow ()] \in SG_D$, d.h. das Attribut r ist auf Klasse a nicht definiert. In diesem Fall widerspricht das OC der Syntax der *F-Logic*, da OCs nur über eigentliche Attribute einer Klasse definiert sind.

(4) AX ist verletzt. Drei Fälle:

(a) UN wird verletzt.
Dieser Fall kann aufgrund der *unique name assumption* nicht eintreten.

(b) NN wird verletzt,
$\exists C \exists A \exists O \forall V: (C[A \Rightarrow ()] \wedge (O : C) \wedge \neg O[A \rightarrow V])$, d.h. $c[r \Rightarrow d] \in SG_D$ und es gibt einen Objekt-ID-Term $o[]$ mit $o : c \in pp_{f_{\mathcal{I}}}$, so dass es keinen Objekt-ID-Term $p[]$ gibt mit $o[r \rightarrow p] \in ob_{f_{\mathcal{I}}}$. Da $o : c$ gilt $o^w \in (c^w)^{\mathcal{I}}$ aufgrund der Konstruktion von $f_{\mathcal{I}}$. Da $c[r \Rightarrow d] \in SG_D$ gilt aufgrund der Konstruktion von K_B $(c^w \sqsubseteq (\leq 1\ [2]r^w) \sqcap (\geq 1\ [2]r^w)) \in \mathcal{T}_K$. Also gibt es genau ein Individuum p^w mit $(o^w, p^w) \in (r^w)^{\mathcal{I}}$ und damit auch ein $o[r \rightarrow p] \in ob_{f_{\mathcal{I}}}$. Ein Widerspruch.

(c) WT wird verletzt. WT besteht aus zwei Axiomen:

$$WT_1 = \{\forall O \forall A \forall V \exists C{:}\ (C[A \Rightarrow ()] \wedge O{:}C) \longleftarrow O[A \rightarrow V]\}$$
$$WT_2 = \{\forall O \forall A \forall V \forall C \forall R{:}\ V{:}R \longleftarrow C[A \Rightarrow R] \wedge O{:}C[A \rightarrow V]\}.$$

Es gibt bei Verletzungen von WT also zwei Falle:

i. WT_1 wird verletzt: Dann gibt es $o[r \rightarrow p] \in ob_{f_\mathcal{I}}$, aber $o[]$ ist nicht Instanz einer Klasse, für die r definiert ist:
$\nexists c[] \in CL_D : (o : c \in pp_{f_\mathcal{I}} \wedge c[r \Rightarrow ()])$.
Da $o[r \rightarrow p] \in ob_{f_\mathcal{I}}$ gilt aufgrund der Konstruktion von $f_\mathcal{I}$: $(o^w, p^w) \in (r^w)^\mathcal{I}$. Für r^w gilt aufgrund der Konstruktion von K_B:

$$(r^w \sqsubseteq \bigsqcup_{(c_i[r \Rightarrow a_i]) \in SG_D} [(1/2 : c_i^w) \sqcap (2/2 : a_i^w)]) \in \mathcal{T}_K.$$

Deshalb gibt es insbesondere ein i mit $o^w \in (c_i^w)^\mathcal{I}$ und $p^w \in (a_i^w)^\mathcal{I}$. Insbesondere gibt es eine Signatur $c_i[r \Rightarrow a_i] \in SG_D$. Nach der Konstruktion von $f_\mathcal{I}$ gilt dann aber: $\{o : c_i, p : a_i\} \subseteq ob_{f_\mathcal{I}}$. Ein Widerspruch.

ii. WT_2 wird verletzt. O.B.d.A. sei die Signatur $a[r \Rightarrow b] \in SG_D$ betroffen. Dann gibt es einen Objekt-ID-Term $o[]$ mit $o : a \in pp_{f_\mathcal{I}}$, $o[r \rightarrow p] \in ob_{f_\mathcal{I}}$ und $p : b \notin pp_{f_\mathcal{I}}$.
Aufgrund der Konstruktion von $f_\mathcal{I}$ gilt: $o^w \in (a^w)^\mathcal{I}$, $p^w \notin b^w$ und $(o^w, p^w) \in (r^w)^\mathcal{I}$. Aufgrund der Konstruktion von K_D gilt:
$\{(a^w \sqsubseteq (\forall r^w|_{[1][2]}.b^w)), (a^w \sqsubseteq (\leq 1\,[2]r^w) \sqcap (\geq 1\,[2]r^w))\} \subseteq \mathcal{T}_K$.
Daher gibt es ein $q^w \in (b^w)^\mathcal{I}$ mit $(o^w, q^w) \in (r^w)^\mathcal{I}$ und aufgrund der Anzahlbeschränkungen gilt $p^w = q^w$. Ein Widerspruch.

Somit ist $f_\mathcal{I}$ eine Instanz von D, womit die Behauptung folgt. ∎

Satz 5.5 (Reduzierbarkeit der logischen Implikation). *Sei D ein* F-Logic-*Schema, das keine Pfadabhängigkeiten enthält, sei τ eine CIC oder OC, τ^w der Widerpart von τ und sei K_D der $\mathcal{DLR}_{reg,ifd}$-Widerpart zu D. Unter der* unique name assumption *gilt dann:*

(1) D hat eine Instanz gdw. K_D erfüllbar ist.

(2) Es gilt $D \models \tau$ gdw. $K_D \models \tau^w$.

Beweis:

(1) „⇒“: Konstruiere aus einer Instanz f von D den Widerpart $\mathcal{I}_f$. Nach Satz 5.3 ist $\mathcal{I}_f$ ein Modell von K_D.
„⇐“: Konstruiere aus einem Modell $\mathcal{I}$ die simulierte Extension $f_\mathcal{I}$. Nach Satz 5.4 ist $f_\mathcal{I}$ eine Instanz von D.

(2) Folgt aus der Äquivalenz von $D \models \tau$ und „$D \cup \{\tau\}$ ist erfüllbar“ in der *F-Logic* einerseits und der Äquivalenz von $K \models \alpha$ mit „$K \cup \alpha$ ist erfüllbar“ für eine Zusicherung α in $\mathcal{DLR}_{reg,ifd}$ andererseits.

∎

5.3.5 Diskussion

In $\mathcal{DLR}_{reg,ifd}$ lässt sich ein Erfüllbarkeitsproblem und damit auch das Problem der logischen Implikation für semantische Bedingungen in der beschriebenen *F-Logic* ohne Pfadabhängigkeiten und unter der *unique name assumption* entscheiden. Pfadabhängigkeiten sind in $\mathcal{DLR}_{reg,ifd}$ nicht darstellbar.

Insgesamt kann $\mathcal{DLR}_{reg,ifd}$ also nicht für die Entscheidung der *F-Logic* mit PFDs durch eine Reduktion herangezogen werden. Zum Versuch einer Erweiterung von $\mathcal{DLR}_{reg,ifd}$ mit dem Ziel, nicht-unäre Pfadabhängigkeiten formulieren zu können, sei auf Abschnitt 5.7 verwiesen.

5.4 Modellierung der F-Logic in DLClass

`DLClass` [KTW01, vgl. Abschnitt 3.4 dieser Arbeit] besitzt lediglich Konzeptkonstruktoren und zwar

- (all f D): *value-restriction*,
- (and D D): *concept intersection* und
- (fd C : $\mathrm{Pf}_1, \cdots, \mathrm{Pf}_n \rightarrow \mathrm{Pf}), k > 1$: Objekte, die in Bezug auf das primitive Konzept C der angegebenen PFD genügen.

Rollenkonstruktoren sind nicht vorgesehen, stattdessen werden durch Verkettung von Attributen Pfade gebildet. Zyklische Terminologien werden explizit ausgeschlossen.

Das Implikationsproblem für `DLClass` ist entscheidbar und DEXPTME-vollständig [KTW01]. Spezialfälle, in denen die linke Seite der PFD einen Präfix der rechten Seite enthält, wobei letztere um höchstens ein Attribut verlängert wird, sind in PTIME entscheidbar [KTW01]. In [KW03] wird ein effizienter Algorithmus zur Entscheidung des *membership problem* mit asymmetrischen fd-Konstrukten vorgestellt.

Wie im vorangegangenen Versuch der Darstellung eines *F-Logic*-Schemas mit vollständiger Signatur und Hierarchie durch eine $\mathcal{DLR}_{reg,ifd}$-Wissensbasis soll auch hier vorgegangen werden. Zu Gunsten einer kurzen Darstellung wird hier jedoch auf die formalen Definitionen verzichtet. Der Widerpart eines jeden *F-Logic*-ID-Terms $x[]$ sei jeweils x^w. Für jede Klasse $a[] \in CL_D$ wird ein atomares Konzept a^w, für jedes Attribut $r[] \in AT_D$ ein Attribut r^w eingeführt.

In `DLClass` können dann durch einen Widerpart ausgedrückt werden: Die Klassenhierarchie HR_D, die Signatur SG_D, aus der Menge der semantischen Bedingungen die CICs und PFDs, nicht jedoch die OCs (dazu fehlt eine inverse Rolle, wobei `DLClass` insbesondere nur *totale* Funktionen als Attribute zulässt):

Sei D ein vollständiges *F-Logic*-Schema. Dann wird der `DLCass`-Widerpart von D wie folgt konstruiert:

- Klassenhierarchie/*class is-a assertions*:

  $$HR_D^w = \{(c^w < d^w) | (c :: d) \in HR_D\}.$$

- Signaturen von Attributen/sklarare Signaturausdrücke:

  $$SG_D^w = \{(c^w < (\text{all}\, a^w\, d^w)) | c[a \Rightarrow d] \in SG_D\}.$$

  Da in `DLClass` alle Attribute totale Funktionen sind, erübrigen sich die im $\mathcal{DLR}_{reg,ifd}$ notwendigen Zusicherungen mit Anzahlbeschränkungen.

- Klasseninklusionsabhängigkeiten:

  $$(c^w < d^w) \in SC_D^w \qquad \text{gdw.} \qquad (c \subset d) \in SC_D$$

- Onto-Abhängigkeiten $c\{a|d\}$: Die Modellierung ist nicht möglich.

- Pfadabhängigkeiten:
  $\{(c^w < (\text{fd } \text{Pf}_1^w, \cdots, \text{Pf}_n^w \rightarrow \text{Pf}_i'^w)) | 1 \leq i \leq k\} \subseteq SC_D^w$ gdw.
  $c(\text{Pf}_1 \cdots \text{Pf}_n \rightarrow \text{Pf}_1' \cdots \text{Pf}_k') \in SC_D$. Für die Widerparte von Pfaden gilt dabei:
  $(.Id)^w = .Id$, $(a.\text{Pf}')^w = a^w.(\text{Pf}')^w$.

Der Widerpart des Schemas D ist dann die `DLClass`-Terminologie

$$\mathcal{T}_D = HR_D^w \cup SG_D^w \cup SC_D^w.$$

Das *Not-Null-Axiom* der *F-Logic* wird durch die Totalität der Attribute in `DLClass` per se erfüllt. Eine Beachtung des *Well-Typedness*-Axioms WT_2 kann in `DLClass` ebensowenig erzwungen werden wie die der *Unique-Name*-Axiome.

Ähnliches gilt für die in Abschnitt 5.4 vorgestellten Varianten von `DLClass` aus [TW01]:

- In $\mathcal{DLFR}$ können mittels Rollen ausgedrückt werden:
  - Signaturen $a[r \Rightarrow b]$ durch $a^w < \forall r^w.b^w, \exists (r^w)^-.a^w < b$
  - CICs $(a \subset b)$ durch $a^w < b^w$
  - OCs $a\{r|b\}$ durch $b^w < \forall (r^w)^-.a^w$
  - PFDs können nicht formuliert werden.

  - Rollen sind nicht funktional und ihre Funktionalität kann nicht erzwungen werden. Daher kann die Funktionalität der Attribute nicht sichergestellt werden.[2] Alternativ kann man Attribute als totale Funktionen auffassen. Dann ist zwar einerseits die Funktionalität der Attribute und die Einhaltung des NN-Axioms gewährleistet, andererseits können dann keine Onto-Abhängigkeiten ausgedrückt werden, da die Funktionen, die Rollen modellieren nicht mit anderen Funktionen wechselwirken dürfen.
  - Das WT_1-Axiom kann wiederum nicht sichergestellt werden.[3]

- Im Falle von $\mathcal{DLFD}$ ergibt sich dasselbe Bild wie bei `DLClass`.
- In $\mathcal{DLFRD}$, der Erweiterung von $\mathcal{DLFR}$ um *dependencies*, können Signaturen, CICs und OCs wie in $\mathcal{DLFR}$ formuliert werden. Zwar können PFDs formuliert werden wie unter `DLClass`, jedoch müssen Attribute, die Rollen modellieren, und die „echten“ Attribute, über die Pfade gebildet werden können, disjunkte Mengen bilden. Folglich darf der Widerpart eines Attributs entweder in einem Pfad des Widerparts einer PFD erscheinen oder er kann im Widerpart einer Signatur oder einer Onto-Abhängigkeit vorkommen. Man kann natürlich für diese Fälle einen aus einem Attribut *und* einer Rolle bestehenden Widerpart vorsehen, nur ist es unmöglich, die notwendige Gleichheit beider zu erzwingen. Somit kann man auch in $\mathcal{DLFRD}$ kein allgemeines *F-Logic*-Schema darstellen.

Es zeigt sich, dass es nicht möglich ist, die beschriebene *F-Logic* unter Berücksichtigung aller drei Klassen von semantischen Bedingungen (CICs, OCs und PFDs) in `DCLass` oder den beschriebenen Varianten dieser *Description Logic* vollständig zu modellieren. Es können allerdings *F-Logic*-Schemata, in deren semantischen Bedingungen keine Onto-Abhängigkeiten in `DLClass`, $\mathcal{DLFD}$ oder $\mathcal{DLFDR}$ modelliert werden.

5.5 Modellierung der F-Logic in $\mathcal{ALCQI}_{reg}$

Die *Description Logic* $\mathcal{ALCQI}_{reg}$ wurde in Abschnitt 3.5 vorgestellt. Im Vergleich zu $\mathcal{DLR}_{reg,ifd}$ fällt insbesondere auf, dass es in $\mathcal{ALCQI}_{reg}$ keine n-ären Relationen und keine Rolleninklusionszusicherungen (in Analogie zu den Relationsinklusionszusicherungen in $\mathcal{DLR}_{reg,ifd}$) gibt. An die Stelle der n-ären Relationen und der regulären Ausdrücke treten in $\mathcal{ALCQI}_{reg}$ Rollen (binäre Relationen).

In direkter Analogie zur Bildung des $\mathcal{DLR}_{reg,ifd}$-Widerparts eines *F-Logic*-Schemas (vgl. Abschnitt 5.3.1) sind in $\mathcal{ALCQI}_{reg}$ sind folgende Modellierungen von $\mathcal{ALCQI}_{reg}$-Widerparten der Komponenten eines *F-Logic*-Schemas D möglich:

[2] Man beachte jedoch Bemerkung 3.1 auf Seite 58.

[3] Aus Bemerkung 3.1 folgt jedoch, dass diese nach entsprechenden Erweiterungen wie im Fall von $\mathcal{DLR}_{reg,ifd}$ erzwungen werden können.

- Klassenhierarchie:

$$HR_D^w = \{(c^w \sqsubseteq d^w) | c :: d \in HR_D\}.$$

- Menge der Signaturen:

$$SG_D^w = \bigcup_{c[a \Rightarrow d] \in SG_D} \{(c^w \sqsubseteq \exists^{\leq 1}\, a^w.d^w), (c^w \sqsubseteq \exists^{\geq 1}\, a^w.d^w), (\exists\, (a^w)^-.c^w)\}.$$

- Klasseninklusionsabhängigkeiten:

$$(c^w \sqsubseteq d^w) \in SC_D^w \qquad \text{gdw.} \qquad (c \subset d) \in SC_D.$$

- Onto-Abhängigkeiten:

$$(d^w \sqsubseteq \neg\exists\, (a^w)^-.\neg c^w) \in SC_D^w \qquad \text{gdw.} \qquad c\{a|d\} \in SC_D.$$

- Pfadabhängigkeiten: $c(\mathrm{Pf}_1 \cdots \mathrm{Pf}_n \rightarrow \mathrm{Pf})$ sind nicht darstellbar.

Unter der *unique name assumption* müssen die UN-Axiome nicht eigens überprüft werden. Das WT_1- sowie das NN-Axiom wird durch die Widerparte der Signaturausdrücke berücksichtigt, das WT_2-Axiom ist nicht modellierbar.

Also ist, wie bereits im Fall von $\mathcal{DLR}_{reg,ifd}$, die *F-Logic* allenfalls unter Ausschluss von Pfadabhängigkeiten in $\mathcal{ALCQI}_{reg}$ modellierbar.

Tatsächlich ist $\mathcal{ALCQI}_{reg}$ ein Fragment von $\mathcal{DLR}_{reg}$ [CG03].[4] Dies kann man durch die Darstellung der klassischen *Description Logic*-Konstrukte durch $\mathcal{DLR}_{reg}$-Ausdrücke nachweisen:

$\mathcal{ALCQI}_{reg}$	$\mathcal{DLR}_{reg}$
$\exists R.C$	$\exists E_R.C$
$\forall R.C$	$\neg\exists E_R.\neg C$
$\exists^{\leq n} P.C$	$(\leq k\, [1](P \sqcap (2/2:\ C)))$
$\exists^{\leq n} P^-.C$	$(\leq k\, [2](P \sqcap (1/2:\ C)))$

Der Ausdruck E_R wird dabei wie folgt gebildet:

R	E_R
P	$P\vert_{[1][2]}$
P^-	$P\vert_{[2][1]}$
$R_1 \sqcup R_2$	$E_{R_1} \sqcup E_{R_2}$
$R_1 \circ R_2$	$E_{R_1} \circ E_{R_2}$
R_1^*	$E_{R_1}^*$
$\mathrm{id}(C)$	$(1/2:\ C)\vert_{[1][2]})$
$(R_1 \sqcup R_2)^-$	$R_1^- \sqcup R_2^-$
$(R_1 \circ R_2)^-$	$R_2^- \circ R_1^-$
$(R_1^*)^-$	$(R_1^-)^*$
$(\mathrm{id}(C))^-$	$\mathrm{id}(C)$

[4] Dabei ist $\mathcal{ALCQI}_{reg}$ ohne die optionalen Erweiterungen aus Tabelle 3.6 gemeint.

Diese Reduktion zeigt, dass $\mathcal{ALCQI}_{reg}$ höchstens genauso aussagekräftig sein kann, wie $\mathcal{DLR}_{reg}$. Daher kann auch die Modellierung eines der *F-Logic* in $\mathcal{ALCQI}_{reg}$ nicht besser erfolgen, als is $\mathcal{DLR}_{reg,ifd}$.

Auch die optionalen Erweiterungen von $\mathcal{ALCQI}_{reg}$ um Differenz eingeschränkter Rollen ($S_1 \setminus S_2$), eingeschränkte *role-value-maps* ($S_1 \subseteq S_2$) und den Durchschnitt von eingeschränkten Rollen ($S_1 \sqcap S_2$) helfen nicht dazu, die Situation bei der Modellierung von *F-Logic*-Schemata zu verbessern. Zwar können mittels der (stark beschränkten) *role-value-maps* Rolleninklusionsbeziehungen für ein Paar atomarer Rollen oder deren Inversen S_1, S_2 Rolleninklusionszusicherungen modelliert werden:

$$C \sqcup \neg C \sqsubseteq (S_1 \subseteq S_2) \qquad \text{entspricht} \qquad S_1 \sqsubseteq S_2.$$

Derartige Zusicherungen reichen jedoch nicht aus, um etwa das WT_2-Axiom modellieren zu können.

Somit ist auch das (erweiterte) $\mathcal{ALCQI}_{reg}$ als Zielformalismus für eine Reduktion des Implikationsproblems für alle drei Klassen von semantischen Bedingungen in der *F-Logic* unzureichend geeignet.

5.6 Verbindungen von Description Logics

In den vorausgegangenen Kapiteln wurde deutlich, dass es prinzipiell möglich ist, Klasseninklusionsabhängigkeiten, Onto-Abhängigkeiten und Pfadabhängigkeiten in verschiedenen *Description Logics* darzustellen. So konnten

- in $\mathcal{ALCQI}_{reg}$ Klasseninklusionsabhängigkeiten, Onto-Abhängigkeiten sowie unäre Schlüssel für Rollen und Konzepte formuliert werden, nicht jedoch Pfadabhängigkeiten,
- in $\mathcal{DLR}_{reg,ifd}$ Klasseninklusionsabhängikeiten, Onto-Abhängigkeiten sowie als Teilmenge der Uniqueness Constraints unäre Schlüsselabhängigkeiten für Relationen, Schlüssel für Klassen und nicht-unäre funktionale Abhängigkeiten für Relationen dargestellt werden, jedoch keine allgemeinen Pfadabhängigkeiten,
- in `DLClass` und $\mathcal{DLFD}$ Klasseninklusionsabhängigkeiten und beliebige Pfadabhängigkeiten (somit auch funktionale Abhängigkeiten und Schlüssel für Konzepte), nicht jedoch Onto-Abhängigkeiten dargestellt werden.

Für jeden der genannten Formalismen ist das Problem der logischen Implikation entscheidbar. In keiner der untersuchten *Description Logics* konnten jedoch alle drei zu berücksichtigenden Klassen von semantischen Bedingungen gemeinsam formuliert werden.

5.6.1 Vereinigungen von Description Logics

Ein naheliegender Ansatz zur Lösung dieses Problems besteht darin, die Ausdrucksmöglichkeiten verschiedener *Description Logic*-Sprachen miteinander zu kombinieren. Das Zusammenfassen zweier *Description Logics* bedeutet dabei, dass man die Konstruktorensätze beider Logiken zusammenlegt und dadurch einen neuen Formalismus, die *Vereinigung* der *Description Logics* (*union of Description Logics*), schafft. In diesem neuen Formalismus kann man von den Ausdrucksmöglichkeiten beider Komponentenformalismen profitieren, indem man die Konstruktoren beider Komponentenformalismen gemeinsam miteinander verwendet.

Leider ergibt sich bei diesem Vorgehen das Problem, dass obwohl die zu vereinigenden Formalismen jeder für sich ein entscheidbares Erfüllbarkeitsproblem und Implikationsproblem haben mögen, sich diese wünschenswerte Eigenschaft jedoch nicht ohne Weiteres auch auf das Ergebnis der Vereinigung beider überträgt. Vielmehr muss in der Regel ein neues Entscheidungsverfahren für diese Vereinigung gefunden werden — was in der Regel mit einem erheblichen Aufwand verbunden ist und häufig auch zu negativen Ergebnissen führt.

Um diesen Aufwand zu vermeiden, kann man versuchen, „kritische“ Konstrukte der zu kombinierenden Ausgangsformalismen getrennt voneinander zu halten. „Kritisch“ sind vor allem solche Konstruktoren, die miteinander dergestalt wechselwirken, dass sich die Komplexität des Entscheidungsverfahrens beträchtlich erhöht. Für *Description Logics* gehören zu diesen „kritischen“ Konstruktoren insbesondere Rollenkonstruktoren, wie die relationale Komposition, der Durchschnitt und das Komplement von Rollen in Verbindung mit Anzahlbeschränkungen und *role-value-maps*.

Könnte man ein Verfahren angeben, das zwei entscheidbare *Description Logics* miteinander kombiniert und dabei die Trennung der „kritischen“ Konstrukte sicherstellt, so wäre dies ein großer Gewinn.

5.6.2 Fusionen von Description Logics

Ein Verfahren, das zwei Description Logics in einer Weise miteinander verbindet, dass „kritische“ Konstrukte voneinander getrennt bleiben, stellt die *Fusion* (*fusion*) von *Description Logics* dar. Diese Methode stammt aus dem Bereich der Modalen Logiken und wurde ursprünglich auf normale Modallogiken angewandt. Baader et al. haben dieses Verfahren zunächst auf bestimmte *Description Logics* ohne Berücksichtigung von ABoxes übertragen [BLSW00]. In einer neueren Arbeit [BLSW02] konnten sie ihre Ergebnisse jedoch auch auf viele *Description Logics* mit Berücksichtigung von ABoxes erweitern.

Die Anforderungen an die zu fusionierenden Komponenten-DLs sind dabei:

(1) Die DLs müssen für Konzepte über alle Bool'schen Operatoren verfügen

$(C_1 \sqcap C_2,\ C_1 \sqcup C_2,\ \neg C_1)$. Dieses erste Kriterium wird beispielsweise von allen $\mathcal{ALC}$-Erweiterungen erfüllt.

(2) Die DLs dürfen keine Individuen (*nominals*) in Konzeptbeschreibungen unterstützen ($\{a\}$).

(3) Die DLs dürfen weder die universale Rolle (Δ^2), noch die Negation (d.h. das Komplement bezüglich Δ^2) von Rollen (R^C) unterstützen.

Für diese Arten von *Description Logics* gelang es den Autoren insbesondere nachzuweisen, dass ihre Fusionen folgende Eigenschaften erhalten, wenn sie bei beiden Komponenten-DLs vorliegen:

(1) Entscheidbarkeit der Konsistenz von ABoxes unter Berücksichtigung einer TBox [BLSW02, Theorem 17];

(2) Entscheidbarkeit der Konzepterfüllbarkeit unter Berücksichtigung einer TBox [BLSW02, Korollar 22];

(3) Entscheidbarkeit der Konsistenz von ABoxes ohne Berücksichtigung einer TBox [BLSW02, Theorem 29];

(4) Entscheidbarkeit der Konzepterfüllbarkeit ohne Berücksichtigung einer TBox [BLSW02, Korollar 34].

Die Fusion $\mathcal{D}_1 \otimes \mathcal{D}_2$ zweier *Description Logics* $\mathcal{D}_1$ und $\mathcal{D}_2$ erlaubt die Kombination der Konstruktoren von $\mathcal{D}_1$ und $\mathcal{D}_2$ in Konzeptbeschreibungen. Um jedoch „kritische“ Konstrukte beider Komponenten-DLs zu trennen, wird in der Fusion die Menge aller verwendeten Rollen in zwei Mengen partitioniert: eine Menge $\mathcal{R}_1$ zum alleinigen Gebrauch mit Konstruktoren der DL $\mathcal{D}_1$ und eine Menge $\mathcal{R}_2$ zum alleinigen Gebrauch mit Konstruktoren aus $\mathcal{D}_2$.

So wird auch in Gegenwart von Rollenhierarchien (bzw. Rolleninklusionszusicherungen in der TBox) für jede Komponenten-DL $\mathcal{D}_i$ eine eigene Hierarchie bezüglich $\mathcal{R}_i$ angenommen, wobei beide Hierarchien strikt voneinander getrennt bleiben.

Damit stellt die Fusion $\mathcal{D}_1 \otimes \mathcal{D}_2$ ein Fragment der Vereinigung von $\mathcal{D}_1$ und $\mathcal{D}_2$ dar. Darin können solche Konzepte definiert werden, die sowohl Konstruktoren aus $\mathcal{D}_1$, als auch Konstruktoren aus $\mathcal{D}_2$ benötigen, wobei jedoch Konstruktoren aus $\mathcal{D}_1$ niemals mit Rollen aus $\mathcal{R}_2$ auftreten und umgekehrt.

Der große Vorteil einer Fusion besteht darin, dass man für das Folgern keiner neuen Methode bedarf, sondern dieses auf das Folgern in den Komponenten-DLs zurückführen kann. Somit bleibt beispielsweise auch die Entscheidbarkeit der Erfüllbarkeits- und Konsistenzprobleme in der Fusion erhalten.

Bei der Vorstellung der DL $\mathcal{DLFR}$ (Abschnitt 3.4) wurde in Bemerkung 3.1 auf die Erweiterbarkeit der Logik $\mathcal{DLFR}$ um *dependencies* aufmerksam gemacht. Die

Begründung für die Erweiterbarkeit war, dass in $\mathcal{DLFRD}$ die Rollen aus $\mathcal{DLFR}$ und die totalen Funktionen aus $\mathcal{DLFD}$ nicht miteinander interagieren und sich ihr Auftreten auf die jeweiligen Konstruktoren der jeweiligen Sprache beschränkt. $\mathcal{DLFRD}$ ist somit ein Beispiel für eine Fusion: $\mathcal{DLFRD} = \mathcal{DLFR} \otimes \mathcal{DLFR}$.

5.6.3 Fusion von $\mathcal{ALCQI}_{reg}$ und $\mathcal{DLFD}$

Nun kann man zur Darstellung des gesamten Satzes an Klassen semantischer Bedingungen eine Fusion von $\mathcal{ALCQI}_{reg}$ und $\mathcal{DLFD}$ bilden.

Die Überprüfung der Voraussetzungen verläuft zufriedenstellend: Sowohl $\mathcal{ALCQI}_{reg}$ als auch $\mathcal{DLFD}$

(1) sind bezüglich der Konzeptbildung Bool'sch abgeschlossen: Beide DLs verfügen über die Konstruktoren $\neg C$und $C_1 \sqcap C_2$ und somit auch über $C_1 \sqcup C_2 = \neg(\neg C_1 \sqcap \neg C_2)$;

(2) erlauben keine Individuen innerhalb von Konzeptbeschreibungen;

(3) unterstützen die universale Rolle weder explizit, noch mittels der Komplementbildung für Rollen.

Also kann man die Fusion $\mathcal{FL} = \mathcal{ALCQI}_{reg} \otimes \mathcal{DLFD}$ bilden und erhält für die Fusion die Entscheidbarkeit der Konsistenz von ABoxes unter Berücksichtigung einer TBox. Da die logische Implikation beider Komponenten in EXPTIME liegt, ist die ihrer Fusion in 2EXPTIME.

In $\mathcal{FL}$ kann man nun sämtliche Klassen von semantischen Bedingungen ausdrücken:

- CICs $a \subset b$ durch Konzeptinklusionszusicherungen $a^w \sqsubseteq b^w$,
- OCs $a\{r|b\}$ durch die Zusicherung $b^w \sqsubseteq \forall (r^w)^-.a^w$ unter Verwendung der Werterestriktion und der inversen Rolle aus der Komponente $\mathcal{ALCQI}_{reg}$,
- PFDs $c(\mathrm{Pf}_1 \cdots \mathrm{Pf}_k \rightarrow \mathrm{Pf}_{(k+1)} \cdots \mathrm{Pf}_n)$ durch $(n-k)$ Zusicherungen

$$\begin{aligned} c^w &\sqsubseteq c^w\{(\mathrm{Pf}_1^w)^=, \ldots, (\mathrm{Pf}_k^w)^= \rightarrow (\mathrm{Pf}_{k+1}^w)^=\} \\ &\vdots \\ c^w &\sqsubseteq c^w\{(\mathrm{Pf}_1^w)^=, \ldots, (\mathrm{Pf}_k^w)^= \rightarrow (\mathrm{Pf}_n^w)^=\} \end{aligned}$$

unter Verwendung des *dependency*-Konstruktors aus der Komponente $\mathcal{DLFD}$.

Leider ergibt sich folgendes Problem: Innerhalb der Widerparte der Pfadbeschreibungen Pf_i könnte eine Rolle s^w auftreten, die auch im Widerpart einer Onto-Zusicherung $a\{s|b\}$ auftritt. In diesem Falle verstößt die Darstellung gegen die Syntax der Fusion $\mathcal{FL}$, da hier s^w sowohl in einem Konstruktor aus $\mathcal{ALCQI}_{reg}$, als auch in einem aus $\mathcal{DLFD}$ stammenden Konstruktor auftritt. Ähnliches wird bei jedem Widerpart einer Signatur $c[s \Rightarrow d]$ geschehen.

Auch wenn man anstelle von $\mathcal{ALCQI}_{reg}$ die DL $\mathcal{DLR}_{reg,ifd}$ zur Bildung der Fusion mit $\mathcal{DLFD}$ heranzieht, ändert sich an diesem Problem nichts (was angesichts der Erkenntnis, dass $\mathcal{ALCQI}_{reg}$ ein Fragment von $\mathcal{DLR}_{reg,ifd}$ ist auch nicht verwundern wird). Für die Fusion von $\mathcal{DLFR}$ und $\mathcal{DLFD}$ wurde dasselbe Problem bereits in Abschnitt 5.4 dargestellt.

Der Ansatz, eine Fusion mehrerer für sich unzureichender *Description Logics* zur Formulierung des Implikationsproblems für die *F-Logic* zu verwenden, ist somit aufgrund der problemimmanenten Interferenz der semantischen Bedingungen gescheitert.

5.7 Erweiterung von $\mathcal{DLR}_{reg,ifd}$ um Inklusionszusicherungen für reguläre Ausdrücke

In Abschnitt 5.3.1.4.3 wurde der Versuch unternommen, Pfadabhängigkeiten in $\mathcal{DLR}_{reg,ifd}$ darzustellen. Der Versuch scheiterte in doppelter Hinsicht:

(1) Der Ausdruck unärer funktionaler Abhängigkeiten in $\mathcal{DLR}_{reg,ifd}$ führt zur Unentscheidbarkeit des Konzepterfüllbarkeitsproblems [CGL01]. Da Pfadabhängigkeiten funktionale Abhängigkeiten verallgemeinern [Wed92], führen somit auch unäre Pfadabhängigkeiten in $\mathcal{DLR}_{reg,ifd}$ zur Unentscheidbarkeit.

(2) Auch bei Beschränkung auf nicht-unäre Pfadabhängigkeiten ist es nicht möglich, diese in einer $\mathcal{DLR}_{reg,ifd}$-Wissensbasis darzustellen. Der Versuch scheitert daran, dass dazu Inklusionszusicherungen für reguläre Ausdrücke benötigt werden, die in der Syntax nicht vorgesehen sind (vgl. Formel 5.8, S. 82).

Um Inklusionszusicherungen zwischen regulären Ausdrücken formulieren zu können, kann versucht werden, den Sprachumfang von $\mathcal{DLR}_{reg,ifd}$ um Inklusionszusicherungen für reguläre Ausdrücke zu erweitern, also Ausdrücke der Form

$$E_1 \sqsubseteq E_2$$

für reguläre Ausdrücke E_1, E_2 in der TBox einer Wissensbasis zu ermöglichen.

Sollte dies bei Erhalt der Entscheidbarkeit gelingen, wäre zumindest eine interessante Teilmenge von semantischen Bedingungen — nämlich nicht-unäre PFDs, OCs und CICs — der *F-Logic* in $\mathcal{DLR}_{reg,ifd}$ darstellbar und entscheidbar.

5.7.1 Die DL-Sprache $\mathcal{DLR}_{reg,ifd,EIA}$

Die *Description Logic*-Sprache $\mathcal{DLR}_{reg,ifd,EIA}$ erweitert $\mathcal{DLR}_{reg,ifd}$ um Inklusionszusicherungen über reguläre Ausdrücke (EIAs). Dies erfolgt mit den Definitionen 5.10 und 5.11. Ein zusätzlicher Konstruktor für reguläre Ausdrücke, die *Komplementbildung über reguläre Ausdrücke*, ist zur Reduktion der logischen Implikation auf die Wissensbasiserfüllbarkeit notwendig.

Definition 5.10 (Syntax von $\mathcal{DLR}_{reg,ifd,EIA}$). *Die Syntax der Sprache $\mathcal{DLR}_{reg,ifd,EIA}$ entspricht derjenigen von $\mathcal{DLR}_{reg,ifd}$. Zusätzlich sind jedoch Zusicherungen der Form*

$$E_1 \sqsubseteq E_2$$

für reguläre Ausdrücke E_1, E_2 in der TBox $\mathcal{T}_K$ einer Wissensbasis erlaubt. Zusicherungen dieser Form heißen Inklusionszusicherungen über reguläre Ausdrücke (EIAs). *Außerdem gibt es für die Bildung regulärer Ausdrücke einen weiteren Konstruktor*

$$\neg E \quad \textit{(Komplementbildung über reguläre Ausdrücke).}$$

Definition 5.11 (Semantik von $\mathcal{DLR}_{reg,ifd,EIA}$). *Die Semantik von $\mathcal{DLR}_{reg,ifd,EIA}$ entspricht der von $\mathcal{DLR}_{reg,ifd}$. Zusätzlich gilt:*

$$(\neg E)^{\mathcal{I}} := (\Delta^{\mathcal{I}})^2 \setminus E^{\mathcal{I}}.$$

Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ erfüllt die Zusicherung

$$E_1 \sqsubseteq E_2$$

gdw.

$$E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}}.$$

Eine Interpretation $\mathcal{I}$ heißt Modell *einer Wissensbasis $K = \langle E_K | \mathcal{T}_K | \mathcal{A}_K | \mathcal{S}_K \rangle$ gdw. $\mathcal{I}$ alle Zusicherungen in $\mathcal{T}_K \cup \mathcal{A}_K \cup \mathcal{S}_K$ erfüllt.*

5.7.2 Entscheidungsverfahren für $\mathcal{DLR}_{reg,ifd,EIA}$

Das Entscheidungsverfahren für $\mathcal{DLR}_{reg,ifd,EIA}$ soll im Wesentlichen auf dem Entscheidungsverfahren für $\mathcal{DLR}_{reg,ifd}$ beruhen.

Das letztgenannte Verfahren wird in Abschnitt 3.3 ausführlich vorgestellt. Es besteht aus vier Stufen:

(1) Reduktion der logischen Implikation in $\mathcal{DLR}_{reg,ifd}$ auf Wissensbasis-Erfüllbarkeit in $\mathcal{DLR}_{reg}$ (Abschnitt 3.3.1)

(2) Reduktion der Erfüllbarkeit einer $\mathcal{DLR}_{reg}$-Wissensbasis auf die $\mathcal{DLR}_{reg}$-Konzepterfüllbarkeit (Abschnitt 3.3.2)

(3) Reduktion des Erfüllbarkeitsproblems in $\mathcal{DLR}_{reg}$ auf das Erfüllbarkeitsproblem in PDL (Abschnitt 3.3.3)

(4) Entscheidungsverfahren für das Erfüllbarkeitsproblem in $CPDL_g$ (Abschnitt 3.3.4)

Im erweiterten Verfahren für $\mathcal{DLR}_{reg,ifd,EIA}$ wird in Schritt (1) der neue Konstruktor $\neg E$ dazu verwendet, den zusätzlichen Fall $K \models E_1 \sqsubseteq E_2$ zu behandeln. Es gilt:

$$K \models E_1 \sqsubseteq E_2 \text{ gdw. } K \cup \{\top \sqsubseteq \exists[1](P \sqcap (2/2 : \exists(E_1 \sqcup \neg E_2).\top))\} \text{ ist unerfüllbar.}$$

Stufe (2) kann direkt in das Entscheidungsverfahren für $\mathcal{DLR}_{reg,ifd,EIA}$ übernommen werden. Eine weitere Modifikation des Entscheidungsverfahrens muss wieder in Stufe (3) erfolgen.

5.7.2.1 Reduktion des Erfüllbarkeitsproblems in $\mathcal{DLR}_{reg,EIA}$ auf das Erfüllbarkeitsproblem in PDL

5.7.2.1.1 Darstellung von EIAs durch eine Implikation Zunächst wird die Transformationsfunktion σ untersucht, unter deren Verwendung eine $\mathcal{DLR}_{reg}$-Wissensbasis in eine $CPDL_g$-Formel übersetzt wird. Bei genauer Betrachtung der Abbildungsvorschrift bemerkt man, dass sowohl Konzeptausdrücke, wie auch Relationen letztendlich auf Formelausdrücke abgebildet werden:

$\sigma(\top_1)$	$= \top_1$	$\sigma(\top_n)$	$= \top_n$
$\sigma(A)$	$= A$	$\sigma(P)$	$= P$
$\sigma(\neg C)$	$= \top_1 \wedge \neg\sigma(C)$	$\sigma(\neg R)$	$= \top_n \wedge \neg\sigma(R)$
$\sigma(C_1 \sqcap C_2)$	$= \sigma(C_1) \wedge \sigma(C_2)$	$\sigma(R_1 \sqcap R_2)$	$= \sigma(R_1) \wedge \sigma(R_2)$
$\sigma(\exists E.C)$	$= \langle\sigma(E)\rangle\sigma(C)$	$\sigma((i/n : C))$	$= T_n \wedge [f_i]\sigma(C)$
$\sigma(\exists[i]R)$	$= \langle f_i^- \rangle\sigma(R)$		
$\sigma((\leq k[i]R))$	$= [f_i^-]_{\leq k}\sigma(R)$		

Dagegen werden reguläre Ausdrücke immer auf Programmausdrücke abgebildet:

$\sigma(\epsilon)$	$\sigma(\top_1)?$
$\sigma(R\|_{[i][j]})$	$f_i^-; \sigma(R)?; f_j$
$\sigma(E_1 \circ E_2)$	$\sigma(E_1); \sigma(E_2)$
$\sigma(E_1 \sqcup E_2)$	$\sigma(E_1) \cup \sigma(E_2)$
$\sigma(E^*)$	$\sigma(E^*)$

Es fällt auf, dass die Übersetzung der „atomaren“ Bestandteile aller regulären Ausdrücke die Translate der Identität ϵ oder der Projektionen von Relationen $\mathbf{R}|_{[i][j]}$ sind, nämlich Programmausdrücke der Form $\sigma(\top_1)?$ bzw. $f_i^-; \sigma(R)?; f_j$.

Bei der Übersetzung der Inklusionszusicherungen für Konzepte und Relationen wird die Inklusionsbeziehung durch eine Implikation ausgedrückt:

$$\begin{aligned}
\sigma(C_1 \sqsubseteq C_2) &= [U](\sigma(C_1) \rightarrow \sigma(C_2)) && (5.9)\\
&= [U]\neg(\sigma(C_1) \wedge \neg\sigma(C_2)) && (5.10)\\
&= \neg\langle U\rangle(\sigma(C_1) \wedge \neg\sigma(C_2)) && (5.11)\\
\sigma(R_1 \sqsubseteq R_2) &= [U](\sigma(R_1) \rightarrow \sigma(R_2)) && (5.12)\\
&= [U]\neg(\sigma(R_1) \wedge \neg\sigma(R_2)) && (5.13)\\
&= \neg\langle U\rangle(\sigma(R_1) \wedge \neg\sigma(R_2)) && (5.14)
\end{aligned}$$

Da $CPDL_g$ für Programme nicht Bool'sch abgeschlossen ist, können über Programme keine Implikationsbeziehungen formuliert werden, wie es im Falle der CICs und RICs für Formeln geschieht. Zwar gibt es den Operator $r_1 \cup r_2$, es fehlt jedoch ein „Komplement“ zum Ausdruck der Implikation. Ebensowenig lässt sich daher eine passende Übersetzung für den regulären Ausdruck $\neg E$ finden.

Man benötigt also eine Logik, die $CPDL_g$ um das Komplement von Programmen erweitert. Eine solche Logik ist mir jedoch leider nicht bekannt.

5.7.2.1.2 Behandlung regulärer Ausdrücke als Rollen Will man ohne eine Modifikation der PDL auskommen und EIAs dennoch wie die anderen beiden Arten von Inklusionszusicherungen durch Implikationen über $CPDL_g$-Formeln ausdrücken, so müssen in $\mathcal{DLR}_{reg,EIA}$ reguläre Ausdrücke durch Rollen (binäre Relationen) ersetzt werden, für die die Konstruktoren $E_1 \circ E_2, E^*, E_1 \cup E_2$ (und $\neg E$)[5] definiert sind.

Der Sprachumfang von $\mathcal{DLR}_{reg,EIA}$ zwingt dann wiederum zu einer Sonderbehandlung dieser binären Relationen, insbesondere zu starken Restriktionen bezüglich der Anwendbarkeit der Konstruktoren, die Bezug auf Relationen nehmen oder Relationen erzeugen.

Dabei sind insbesondere die Anzahlbeschränkungen $(\leq k\ [i]R)$ in Verbindung mit der konzeptbasierten Relationskonstruktion $(i/n : C)$ problematisch. Durch die Verbindung einer unqualifizierten Anzahlbeschränkung, einer konzeptgebundenen Relationskonstruktion und einer Relationsinklusionszusicherung lässt sich bereits in $\mathcal{DLR}_{reg}$ eine qualifizierte Anzahlbeschränkung formulieren:

$$(\leq k\ [i]\ R.C) \qquad \text{wird ersetzt durch} \qquad (\leq k\ [i]\ (R \sqcap (i/n :\ C))),$$

wobei $n = \text{arity}(R), i \leq n$.

[5] Tatsächlich bedarf es bei Darstellung regulärer Ausdrücke als Rollen keines expliziten Komplement-Konstruktors für Rollen. Auch ohne einen solchen ist die Konstruktion eines echten Rollenkomplements möglich, siehe weiter unten auf S. 105.

Von qualifizierten Anzahlbeschränkungen ist jedoch bekannt, dass ihre Anwendung auf allgemeine Rollen, die unter Verwendung von Rollenkonstruktoren wie Komposition und Durchschnitt gebildet werden, zur Unentscheidbarkeit des Erfüllbarkeitsproblems führt (vgl. Abschnitt 4.4.4). Gleiches gilt auch für die Anwendung qualifizierter Anzahlbeschränkungen auf Rollen mit Komposition, Vereinigung und Inversion, die in $\mathcal{DLR}_{reg,EIA}$ gleichfalls darstellbar sind: Das Komplement für Rollen (resp. reguläre Ausdrücke)[6] zusammen mit dem Durchschnitt von Rollen ergibt die Bool'sche Abgeschlossenheit für Rollen und damit die Darstellbarkeit der Vereinigung von Rollen. Inverse Rollen sind durch Projektionen darstellbar, $R^- = R|_{[2][1]}$. Um die Unentscheidbarkeit des Erfüllbarkeitsproblems zu Vermeiden, müssen die Rollen innerhalb von Anzahlbeschränkungen auf solche beschränkt werde, die mittels „unkritischer" Konstruktoren erzeugt werden. Dabei ist auch die Möglichkeit zu berücksichtigen, dass mittels des Projektionskonstruktors n-ären Relationen mit Rollen verknüpft und somit „kritische" Relationskonstruktoren „importiert" werden können.

Die Verwendung transitiver Rollen in qualifizierten Anzahlbeschränkungen ist bereits für $\mathcal{ALC}_{trans}$ als unentscheidbar bekannt. Transitive Rollen lassen sich in $\mathcal{DLR}_{reg,EIA}$ jedoch mittels EIAs der Form $R^* \sqsubseteq R$ oder $R \circ R \sqsubseteq R$ erzeugen.

Weitere Probleme erwachsen im Zusammenhang mit ϵ, dem regulären Ausdruck der Identität ϵ. Diesem Ausdruck entspricht dann nicht etwa $\top_2$, sondern $\Delta \times \Delta$, also die binäre Universalrelation. Damit ist mittels des $(\neg R)$-Konstruktors die Darstellung eines „echten" Komplements binärer Relationen möglich, indem man mittels einer RIC die Interpretation von $\top_2$ durch $\Delta \times \Delta$ erzwingt:

$$\epsilon \sqsubseteq \top_2.$$

Gemäß der Semantik von $\neg R$ ergibt sich $(\neg R)^{\mathcal{I}} = \top_2^{\mathcal{I}} \setminus R^{\mathcal{I}} = (\Delta^2)^{\mathcal{I}} \setminus R^{\mathcal{I}} = (R^c)^{\mathcal{I}}$.

Wir haben bereits zur Kenntnis genommen, dass sich in $\mathcal{DLR}_{reg}$ qualifizierte Anzahlbeschränkungen darstellen lassen. Nun kann man mit der EIA $(\epsilon \sqsubseteq \top_2) \in \mathcal{T}$ durch Zusicherungen der Form $C \sqsubseteq (\leq k\ [2]P \sqcup \neg P.C)$ die Anzahl der Individuen jedes beliebigen Konzepts C — und somit insbesondere auch die Größe der Domäne (für $C \equiv \top$) — beschränken.

Die Präsenz derartiger Zusicherungen in der DL verletzt jedoch die Voraussetzungen für die Reduktion von beliebigen Anzahlbeschränkungen (bzw. *graded modalities*) auf funktionale Anzahlbeschränkungen (bzw. lokalen Determinismus) in der äquivalenten PDL durch Reifikation [Gia95, Lemma 18, siehe auch Diskussion am Ende von Kapitel 4]. Somit wird der Beweis der Entscheidbarkeit des Erfüllbarkeitsproblems in nichtig. De Giacomo beschreibt in [Gia95] diese Komplikationen am Beispiel der DL $\mathcal{CINB}$ und der PDL $\mathcal{DINB}$. Inwiefern sich dies auf den komplizierteren, hier vorliegenden Fall auswirkt, konnte im Rahmen dieser Arbeit leider nicht untersucht werden.

[6] Siehe S. 104, Fußnote 5.

Die aufgeführten Probleme, die durch die Darstellung regulärer Ausdrücke durch Rollen entstehen, lassen folgende Schlüsse zu:

(1) Um den aufgezeigten Problemen zu begegnen, muss $\mathcal{DLR}_{reg,EIA}$ vollständig neu konzipiert werden. Die Verwendung allgemeiner Relationen und Rollen (resp. regulärer Ausdrücke) in den Konstruktoren muss dazu sorgsam durchdacht werden, um die Entscheidbarkeit des Erfüllbarkeitsproblems in der neuen DL zu gewährleisten. Insgesamt ist fraglich, ob der so zu erhaltende Sprachumfang ausreicht, um die Darstellung von Implikationsproblemen für die *F-Logic* zu ermöglichen.

(2) Es ist ein offenes Problem, ob eine *Description Logic* mit dem gewünschten Sprachumfang überhaupt entscheidbar ist. Aus der gesichteten Literatur ist mir keine *Description Logic*-Sprache bekannt, die vergleichbare Ausdrucksmöglichkeiten bietet.

5.7.3 Weitere Zweifel an der Entscheidbarkeit von $\mathcal{DLR}_{reg,ifd}$ mit EIAs

Weitere grundsätzliche Zweifel an der Möglichkeit, $\mathcal{DLR}_{reg,ifd}$ um EIAs zu erweitern und dabei die Entscheidbarkeit des Erfüllbarkeitsproblems zu erhalten, ergeben sich aus der Literatur zu generalisierten Rolleninklusionszusicherungen (gRIAs). Dabei ist die Beobachtung, dass sich mittels generalisierter Rolleninklusionszusicherungen *role-value-maps* darstellen lassen von Bedeutung [HS03]:[7]

$$R \circ S \sqsubseteq T \Leftrightarrow \text{ jedes Individuum ist Instanz von } (RS \subseteq T) \Leftrightarrow \top \sqsubseteq (RS \subseteq T)$$

für *atomare* Rollen (resp. reguläre Ausdrücke) R, S, T.

Mit Hilfe der EIAs in $\mathcal{DLR}_{reg,ifd,EIA}$ können jedoch weit allgemeinere Formen generalisierter Rolleninklusionszusicherungen formuliert werden. Für binäre Relationen R_i, S_i ist es nämlich möglich, gRIAs der Form

$$R_1 \circ \ldots \circ R_m \sqsubseteq S_1 \circ \ldots \circ S_n$$

für *beliebige* Rollen R_i, S_i folgendermaßen in einer EIA darzustellen:

$$R_1|_{[1][2]} \circ \ldots \circ R_m|_{[1][2]} \sqsubseteq S_i|_{[1][2]} \circ \ldots \circ S_n|_{[1][2]}.$$

Dies ist äquivalent mit einer Zusicherung unter Beteiligung eines RVM-Konstruktors:

$$\top \sqsubseteq (R_1 \cdots R_m \subseteq S_1 \cdots S_n).$$

[7] Zur Definition von Syntax und Semantik der *role-value-maps* siehe Abschnitt 4.4.6, S. 72.

Nebel und Smolka merken an, dass $\mathcal{ALC}$ mit RVMs nur entscheidbar ist, solange die Attribute in den Pfaden der RVMs funktional bleiben. Mehrwertige Attribute führen zur Unentscheidbarkeit des Enthaltenseinsproblems [NeS91]. Nun ist $\mathcal{DLR}_{reg,ifd,EIA}$ eine Erweiterung von $\mathcal{ALC}$ (vgl. dazu Abschnitt 5.5). Zwar treten bei der Darstellung der Implikation in *F-Logic* ausschließlich funktionale Rollen in EIAs auf (vgl. Abschnitt 5.3.1), es bleiben jedoch Zweifel, ob die Entscheidbarkeit des Erfüllbarkeitsproblems in der mit EIAs über funktionale Rollen erweiterten *Description Logic* erhalten bleibt.

Horrocks und Sattler weisen in [HS03] die Unentscheidbarkeit von $\mathcal{SH}^+\mathcal{IQ}$ nach. Dazu reduzieren sie das unentscheidbare Domimo-Problem auf das Erfüllbarkeitsproblem in $\mathcal{SH}^+\mathcal{IQ}$. $\mathcal{SH}^+\mathcal{IQ}$ entspricht der um bestimmte Klassen von Rolleninklusionszusicherungen erweiterten *Description Logic* $\mathcal{ALCIQ}$: Die Sprache verfügt über die Konstruktoren $\neg C, C_1 \sqcap C_2, \exists R.C, \forall R.C, (\geq n\, R.C), (\leq n\, R.C), R^-$. Dabei sind die Rollen in qualifizierten Anzahlbeschränkungen auf einfache Rollen, also solche, die nicht von anderen Rollen abhängen, beschränkt. Die TBox besteht aus Zusicherungen der Form $C_1 \sqsubseteq C_2, R_1 \sqsubseteq R_2, R_1 \circ R_2 \sqsubseteq R_1, R_1 \circ R_2 \sqsubseteq R_2$). Daher ist $\mathcal{SH}^+\mathcal{IQ}$ offensichtlich ein Fragment von $\mathcal{DLR}_{reg,ifd,EIA}$. Somit ist zu vermuten, dass auch $\mathcal{DLR}_{reg,ifd,EIA}$ unentscheidbar ist.

Es gibt weitere Unentscheidbarkeitsergebnisse für *Description Logic*-Sprachen mit generalisierten Rolleninklusionszusicherungen. So weist Wessel die Unentscheidbarkeit von $\mathcal{ALC}_{\mathcal{RA}\ominus}$ nach [Wes00]. Diese *Description Logic* umfasst die Konstruktoren $C_1 \sqcap C_2, \top, \bot, \neg C, \forall R.C, \exists R.C$. TBox-Zusicherungen haben die Form $C_1 \sqsubseteq C_2$ oder $R_1 \circ R_2 \sqsubseteq R_3 \sqcup \ldots \sqcup R_n$, wobei die $R_3, \ldots, R_n$ disjunkt sind. Bereits Rolleninklusionszusicherungen der Form $R_1 \circ R_2 \sqsubseteq R_3$ führen hier zur Unentscheidbarkeit des Erfüllbarkeitsproblems. Durch zusätzliche Beschränkungen für die Rolleninklusionszusicherungen kann zwar Entscheidbarkeit erreicht werden, diese geht jedoch wieder verloren, sobald man die Logik um unqualifizierte Anzahlbeschränkungen erweitert. Auch $\mathcal{ALC}_{\mathcal{RA}\ominus}$ ist ein Fragment von $\mathcal{DLR}_{reg,ifd,EIA}$.

Bereits angesprochen wurde die Unentscheidbarkeit von $\mathcal{SHIF}$ — das ist $\mathcal{ALC}$ erweitert um transitive und inverse Rollen, Rollenhierarchien für primitive Rollen und unqualifizierte funktionale Beschränkungen für Rollen — wenn man innerhalb von Anzahlbeschränkungen transitive Rollen zulässt [HST99, vgl. Abschnitt 4.4.4]. Transitive Rollen sind solche, für die $R \circ R \sqsubseteq R$ gilt.

5.7.4 Ergebnis des Erweiterungsversuchs

Die in den Abschnitten 5.7.2 und 5.7.3 zusammengetragenen Indizien sprechen dafür, dass eine Erweiterung von $\mathcal{DLR}_{reg,ifd}$ um Inklusionszusicherungen für reguläre Ausdrücke oder eine Ersetzung der regulären Ausdrücke durch Rollen sehr kompliziert und höchstwahrscheinlich auch unentscheidbar ist.

Damit ist der Versuch der Ermöglichung der Formulierung von nicht-unären Pfadabhängigkeiten in einer Erweiterung von $\mathcal{DLR}_{reg,ifd}$ gescheitert.

Kapitel 6

Zusammenfassung und Ausblick

In diesem abschließenden Kapitel werden zunächst die Resultate dieser Arbeit zusammengefasst dargestellt. Anschließend werden mögliche Ansatzpunkte und Fragestellungen für eine weitergehende Bearbeitung des Themas aufgewiesen.

6.1 Ergebnisse

Ausgangspunkt für die vorliegende Arbeit war die Frage nach der Entscheidbarkeit des Problems der generellen logischen Implikation für Pfadabhängigkeiten, Klasseninklusionsabhängigkeiten und Onto-Abhängigkeiten im objektorientierten Datenmodell der *F-Logic* nach Biskup und Polle [BP03], für das ein vollständiges und korrektes System von Inferenzregeln bekannt ist.

Nach einer Darstellung der *F-Logic* und der Inferenzregeln wurde eine weitere Klasse von Formalismen eingeführt, die *Description Logics*. Diese Logiken finden breite Anwendung im Bereich der formalen Beschreibung objektorientierter Datenmodelle. Aus der großen Anzahl von in der Literatur beschriebenen *Description Logic*-Sprachen wurden drei Vertreter aussagekräftiger *Description Logic*-Sprachfamilien ausgewählt und vorgestellt: $\mathcal{DLR}_{reg,ifd}$ ermöglicht die direkte Darstellung n-ärer Relationen und deren Anordnung in Relationshierarchien und bietet Konstrukte zum Ausdruck von funktionalen und Schlüsselabhängigkeiten. `DLClass` und deren Abkömmlinge $\mathcal{DLFD}$, $\mathcal{DLFR}$ und $\mathcal{DLFRD}$ verfügen insbesondere über einen Konstruktor zur Beschreibung von generellen Pfadabhängigkeiten. $\mathcal{ALCQI}_{reg}$ stellt eine ausdrucksstarke Erweiterung der „Standard“-*Description Logic* $\mathcal{ALC}$ dar. Die vorgestellten Formalismen haben allesamt entscheidbare Erfüllbarkeits- und Implikationsprobleme, von denen bekannt ist, dass sie EXPTIME-vollständig sind. Die Entscheidungsverfahren nutzen Tableauverfahren oder die Verwandtschaft der *Description Logics* zu einer weiteren Gruppe von Logiken, den *propositional dynamic logics* (PDLs). Anhand des Entscheidungsverfahrens für die logische Implikation in $\mathcal{DLR}_{reg,ifd}$ wurden wichtige Methoden, wie die Internalisierung von ABox und TBox,

sowie die Reifikation von Relationen, zur Reduktion von komplexen *Description Logics* auf einfachere Varianten und schließlich auf eine PDL vorgestellt. Für letztere wurde ein Entscheidungsverfahren skizziert.

Anschließend wurde vom Stand der Forschung im Bereich der Entscheidbarkeit der angesprochenen Klassen von semantischen Bedingungen in objektorientierten Datenmodellen berichtet. Dazu wurden Ergebnisse aus der Literatur zum relationalen Datenmodell, zu einem graph-basierten Datenmodell und insbesondere zu *Description Logics* zusammengetragen.

Von den genannten Formalismen erschienen die *Description Logics* als geeignete Kandidaten für eine Reformulierung des Implikationsproblems der *F-Logic*. Es wurde versucht, dieses Problem auf das Implikationsproblem in den bereits vorgestellten *Description Logic*-Sprachen $\mathcal{DLR}_{reg,ifd}$, $\mathcal{DLFR}/\mathcal{DLFRD}$ und $\mathcal{ALCQI}_{reg}$ zu reduzieren. Die Reduktion gelang in keinem der Fälle vollständig. Der Misserfolg liegt in der Kombination der in den einzelnen *Description Logic*-Sprachen verfügbaren Konzept- und Rollenkonstruktoren begründet. Ein Hauptproblem besteht dabei in der Notwendigkeit, einerseits Attribute als funktionale Rollen darstellen und inverse Rollen zur Abbildung von Onto-Abhängigkeiten bilden zu können, andererseits mittels der Komposition von Rollen Pfadfunktionen bilden und diese ggf. in eine Rollenhierarchie einbinden zu können. Diese beiden Bereiche werden jedoch in *Description Logics* voneinander getrennt, um das Erfüllbarkeitsproblem entscheidbar zu halten.

Eine Mischung dieser Bereiche führt der Literatur zufolge meist zur Unentscheidbarkeit des Erfüllbarkeitsproblems und damit auch der logischen Implikation.

Ein Versuch, durch eine Erweiterung von $\mathcal{DLR}_{reg,ifd}$ in diesem Formalismus zumindest nicht-unäre Pfadabhängigkeiten darstellen zu können, scheiterte daran ebenso wie der Versuch, durch eine Fusion von zwei sich bezüglich des Konstruktorumfangs einander passend ergänzenden *Description Logics* doch noch alle drei Klassen von semantischen Bedingungen gemeinsam formulieren zu können.

Als die „kritischen“ Klassen von semantischen Bedingungen bezüglich der Darstellbarkeit in *Description Logics* erwiesen sich die Onto- und Pfadabhängigkeiten, während Klasseninklusionsabhängigkeiten immer problemlos darstellbar waren.

Zwar ließen sich in keiner der vorgestellten *Description Logics* alle drei Klassen von semantischen Bedingungen formulieren, allerdings lassen sich bestimmte Teilmengen der *Uniqueness Constraints* gemeinsam mit Klasseninklusionsabhängigkeiten und teilweise auch mit Onto-Abhängigkeiten gemeinsam darstellen. Die Ergebnisse dazu sind in Tabelle 6.1 zusammengestellt.

Die ursprüngliche Frage nach der Entscheidbarkeit der logischen Implikation von semantischen Bedingungen in der *F-Logic* konnte in dieser Arbeit jedoch leider nicht beantwortet werden.

sem. Bedingungen		*F-Logic*	$\mathcal{DLR}_{reg,ifd}$	DLClass	$\mathcal{DLRD}$	$\mathcal{ALCQI}_{reg}$
UCs	unäre Schlüssel	+	+	+	+	+
	generelle Schlüssel	+	+	+	+	(+)[‡]
	unäre FDs	+	unentscheidbar	+	+	–
	nicht-unäre FDs	+	+	+	+	–
	PFDs	+	–	+	+	–
OCs		+	+	–	(+)[†]	+
INDs	CICs	+	+	+	+	+

Tabelle 6.1: Darstellbarkeit von semantischen Bedingungen

Anmerkungen zu Tabelle 6.1:

[†] In $\mathcal{DLRD}$ lassen sich zwar PFDs und OCs gemeinsam formulieren, jedoch dürfen diese keine gemeinsamen Attribute aufweisen.

[‡] In $\mathcal{ALCQI}_{reg}$ lassen sich generelle Schlüssel darstellen, wenn man die Sprache um *identification assertions* erweitert.

6.2 Ausblick

Da das anstehende Problem, die Frage nach der Entscheidbarkeit der logischen Implikation von CICs, OCs und PFDs in der *F-Logic* weiterhin offen bleibt, stellt sich die Frage nach Ansatzpunkten zur weiteren Bearbeitung des Problems.

Für die Frage der Entscheidbarkeit ist insbesondere von Interesse, inwiefern die Pfadabhängigkeiten zu den Schwierigkeiten bei der Reformulierung des Problems in *Description Logics* beitragen. Die für die Pfadabhängigkeiten notwendige Verkettung funktionaler Rollen scheint mit *feature agreements* und *role-value-maps* verwandt zu sein; beide sind im Rahmen der *Description Logics* als „kritische“ Konstrukte bekannt.

Während bislang zahlreiche Rollenkonstruktoren für problematisch befunden und ihre nähere Untersuchung als vermeintlich aussichtslos gemieden wurde, gibt es in jüngerer Zeit zunehmend Arbeiten, die sich mit Rollenkonstruktoren, generellen Rolleninklusionsabhängigkeiten und Rollenhierarchien in *Description Logics* beschäftigen. Beispiele für solche Arbeiten sind [Tob00, Gra01, Gra02, Baa03, Don03, HS03].

Auch die Wechselwirkung des zum Ausdruck von Onto-Abhängigkeiten notwendigen Inverse-Rolle-Konstruktors mit den Pfadbeschreibungen scheint ein guter Ansatzpunkt für weitergehende Betrachtungen zu sein.

Während Fusionen aufgrund des möglichen gemischten Auftretens von Attributen sowohl in OCs als auch in PFDs ausscheiden, bleibt die Vereinigung von *Description Logics* zur Lösung des Problems weiterhin möglich. Dieser Fall kommt jedoch der Schaffung einer neuen *Description Logic* gleich.

Anhang A

Abkürzungsverzeichnis

ABox	assertion box	
bzw.	beziehungsweise	
CIC	class inclusion constraint	Konzept-/Klasseninklusionszusicherung
CPDL	Converse PDL	
d.h.	das heißt	
Def.	Definition	
DL	description dogic	
FD	functional dependency	funktionale Abhängigkeit
FL	F-Logic/Frame Logic	
Hrsg.	Herausgeber	
gdw.	genau dann wenn	
ggf.	gegebenenfalls	
i.d.R.	in der Regel	
ID	identification dependency	
INC	inclusion dependency	Inklusionsabhängigkeit
o.B.d.A	ohne Beschränkung der Allgemeinheit	
OC	onto-constraint	Onto-Abhängigkeit
OODM	objektorientiertes Datenmodell	
PDL	propositional dynamic logic	
PE	path equation	
PFD	path functional dependency	Pfadabhängigkeit
RDM	relationales Datenmodell	
RIA	role inclusion axiom	identisch mit RIC
RIC	class inclusion constraint	Rollen-/Relationsinklusionszusicherung
RVM	role-value-map	
S.	Seite	
s.o.	siehe oben	
s.u.	siehe unten	
TBox	terminology box	Terminologie
usw.	und so weiter	
z.B.	zum Beispiel	

Tabellenverzeichnis

3.1 Konstruktoren von $\mathcal{DLR}_{reg,ifd}$ 36
3.2 Abkürzende Notationsweisen für $\mathcal{DLR}_{reg,ifd}$ 37
3.3 Interpretationsfunktion für $\mathcal{DLR}_{reg,ifd}$ 38
3.4 Translation $\mathcal{DLR}_{reg} \rightarrow CPDL_g$ 52
3.5 Syntax und Semantik von $\mathcal{DLF}$ und Erweiterungen 57
3.6 Syntax und Semantik von $\mathcal{ALCQI}_{reg}$ 61

6.1 Ergebnisüberblick 110

Literaturverzeichnis

[Baa03] F. Baader. Restricted role value maps in a description logic with existential restrictions and terminological cycles. Proc. of the 2003 International Workshop on Description Logics (DL2003). CEUR Workshop Proceedings 81, 2003. (`http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-81/`).

[BDNS94] M. Buchheit, F. M. Donini, W. Nutt, A. Schaerf. Terminological systems revisited: Terminology = Schema + Views. Proceedings of 1st Workshop KRDB'94, RWTH Aachen, 1994.

[BFW97] P. Buneman, W. Fan, S. Weinstein. The decidability of some restricted implication problems for path constraints. Technical Report MS-CIS-97-15, Department of Computer and Information Science, University of Pennsylvania, 1997. (`http://citeseer.ist.psu.edu/33603.html`).

[BFW00] P. Bunemann, W. Fan, S. Weinstein. Query optimization for semistructured data using path constraints in a deterministic data model. Lecture Notes in Computer Science 1949: 208–223, 2000.

[BFW03] P. Bunemann, W. Fan, S. Weinstein. Interaction between path and type constraints. ACM Transactions on Computational Logic 4 (4): 530–577, 2003.

[Bis95a] J. Biskup. Database schema design theory: achievements and challenges. Proc. 6th Int. Conf. on Information Systems and Management of Data. LNCS 1006: 14–44, 1995.

[Bis95b] J. Biskup. Grundlagen von Informationssystemen. Vieweg-Verlag 1995.

[BBGR89] A. Borgida, R. J. Brachmann, D. L. McGuinness, L. A. Resnick. CLASSIC: A structural data model for Objects. In: J. Clifford, B. G. Lindsay, D. Maier (Hrsg.). Proc. of the 1989 ACM SIGMOD Int. Conf. on Management of Data: 58–67, ACM Press, 1989.

[BLSW00] F. Baader, C. Lutz, H. Sturm, F. Wolter. Fusions of description logics. In Proc. of DL 2000: 21–30. CEUR Electronic Workshop Proceedings, 2000. (`http://ceur-ws.org/Vol-33/`).

[BLSW02] F. Baader, C. Lutz, Sturm, F. Wolter. Fusions of Description Logics and Abstract Description Systems. Journal of Artificial Intelligence Research (JAIR) 16: 1–58, 2002.

[BN03] F. Baader, W. Nutt: Basic description logics. In: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider. The Description Logic Handbook: Theory, Implementation and Applications. Chapter 2: 47–100, Cambridge University Press, 2003.

[Bor95] A. Borgida. Description logics in data management. IEEE Trans. on Knowledge and Data Engineering 7(5): 671–682, 1995.

[BP00] J. Biskup, T. Polle. Decomposition of Database Classes under Path Functional Dependencies and Onto Constraints. In: K.-D. Schewe, B. Thalheim (Hrsg.): FoIKS 2000, LNCS 1762: 31–49, Springer Verlag, 2000.

[BP01] J. Biskup, T. Polle. Decomposition of object-oriented database schemas. Annals of Mathematics and Artificial Intelligence 33: 119–155, 2001.

[BP03] J. Biskup, T. Polle. Adding inclusion dependencies to an object oriented data model with uniqueness constraints. Acta Informatica 39: 391–449, 2003.

[Bra79] R. J. Brachman. On the epistemological status of semantic networks. In: N. V. Findler (Hrsg.). Associative Networks, Representation and use of knowledge by computers: 3–50, Academic Press, New York 1979.

[BW97] A. Borgida, G. Weddell. Adding uniqueness constraints to description logics. In: Proc. of DOOD'97: 85–102, 1997.

[Cal96] D. Calvanese. Unrestricted and finite model reasoning in class-based representation formalisms. PhD-Thesis, Università degli Studi di Roma „La Sapienzia“, 1996. (`http://www.dis.uniroma1.it/pub/calvanes/thesis.ps.gz`).

[CCGL02] A. Calì, D. Calvanese, G. De Giacomo, M. Lenzerini. A formal framework for reasoning on UML class diagrams. In: Proc. of the ISMIS 2002 (LNCS 2366): 503–513, Springer 2002.

[CG03] D. Calvanese, G. De Giacomo. Expressive description logics. In: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider. The Description Logic Handbook: Theory, Implementation and Applications. Chapter 5: 184–225, Cambridge University Press, 2003.

[CGL95] D. Calvanese, G. De Giacomo, M. Lenzerini. Structured objects: Modeling and reasoning. Proc. of DOOD'95. LNCS 1013: 229–246. Springer Verlag, 1995.

[CGL98] D. Calvanese, G. De Giacomo, M. Lenzerini. On the decidability of query containment under constraints. In Proc. of PODS'98: 149–159, 1998.

[CGL00] D. Calvanese, G. De Giacomo, M. Lenzerini. Keys for free in description logics. Proc. of the 2000 Int. Workshop on Description Logics (DL 2000): 79–88, 2000.

[CGL01] D. Calvanese, G. De Giacomo, M. Lenzerini. Identification constraints and functional dependencies in description logics. IJCAI 2001: 155–160.

[CGL02a] D. Calvanese, G. De Giacomo, M. Lenzerini. Description logics: Foundations for class-based knowledge representation. In: Proc. of 17th IEEE Symposium on Logic in Computer Science (LICS 2002): 359–370, 2002.

[CGL02b] D. Calvanese, G. De Giacomo, M. Lenzerini. Description logics for information integration. In: A. Kakas, F. Sadri (Hrsg.). Logic Programming and Beyond (Essays in honour of Robert A. Kowalski). LNCS 2408: 41–60, Springer 2002.

[CGLNR98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. Description logic framework for information integration. In Proc. KR'98: 2–13, 1998.

[CGNL99] D. Calvanese, G. De Giacomo, D. Nardi, M. Lenzerini. Reasoning in expressive description logics. In: A. Robinson, A. Voronkov (Hrsg.). Handbook of automated reasoning. Elsevier Science Publishers B.V.: 1581–1634, 1999.

[CLN94] D. Calvanese, M. Lenzerini, D. Nardi. A unified framework for class based representation formalisms. In: J. Doyle, E. Sandewall, P. Torasso (Eds.). Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94): 109–120, Morgan Kaufmann, 1994.

[CLN99] D. Calvanese, M. Lenzerini, D. Nardi. Unifying class-based representaion formalisms. Journal of Artificial Intelligence Research 11: 199–240, 1999.

[CV85] A.K. Chandra, M. Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. SIAM J. on Computing, 14(3): 671–677, 1985.

[DLNN91] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt. The complexity of concept languages. Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR'91): 151–162, Los Altos 1991.

[DLNS94] F. Donini, M. Lenzerini, D. Nardi, A. Schaerf. Deduction in concept languages: From subsumption to instance checking. Journal of Logic and Computation 4(4): 423–452, 1994.

[DLNS96] F. Donini, M. Lenzerini, D. Nardi, A. Schaerf. Reasoning in description logics. In: G. Brewka (Hrsg.). Foundation of Knowledge Representation: 191–236. CSLI-Publications, 1996.

[Dob96] G. Dobbie. Towards normalization in object-oriented databases. Technical report CS-TR-96/16, Victoria University of Wellington, 1996.

[Don03] F. M. Donini. Complexity of Reasoning. In: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider. The Description Logic Handbook: Theory, Implementation and Applications. Chapter 3: 101–141, Cambridge University Press, 2003.

[FL79] M. J. Fischer, R. E. Ladner. Propositional dynamic logic of regular programs. J. of Computer and System Sciences, 18(2): 194–211, 1979.

[Gia95] G. De Giacomo. Decidability of class-based knowledge representation formalisms. Ph.D. thesis, Dip. di Inf. e Sist., Univ. di Roma „La Sapienza“, 1995.

[GL94] G. de Giacomo, M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In Proc. of AAAI-94: 205–212, AAAI Press/The MIT Press, 1994.

[GL94b] G. De Giacomo, M. Lenzerini. Description Logics With Inverse Roles, Functional Restrictions, and N-ary Relations. In: C. MacNish, L.M. Pereira, D. Pearce. Logics in Artificial Intelligence: 332–346, Springer-Verlag, 1994. (`ftp://www.dis.uniroma1.it/pub/degiacomo/jelia94.ps.gz/degiacomo94description.ps.gz`).

[GL94c] G. De Giacomo, M. Lenzerini. Converse, local determinism, and graded nondeterminism in propositional dynamic logics. Technical Report 11-94, Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma „La Sapienzia“, 1994.

[GL95] G. de Giacomo, M. Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In Proc. of IJCAI-95: 801–807, 1995.

[GL95b] G. De Giacomo, M. Lenzerini. PDL-based framework for reasoning about actions. LNAI 992: 103–114, 1995.

[GL96] G. De Giacomo, M. Lenzerini. TBox and ABox Reasoning in expressive Description Logics. In: L. C. Aiello, S. C. Shapiro (Hrsg.): Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-9): 316–327, Morgan Kaufmann, Los Altos 1996.

[Gra01] F. Grandi. On expressive number restrictions in description logics. Proc. International Workshop on Description Logics (DL 2001): 56–65, 2001.

[Gra02] F. Grandi. On expressive description logics with composition of roles in number restrictions. Proc. LPAR 2002, LNAI 2514: 202–215, Springer-Verlag, Berlin 2002.

[Har01] S. Hartmann. On the implication problem for cardinality constraints and functional dependencies. Annals of Mathematics and Artificial Intelligence 33: 253–307, Kluwer Academic Publishers, 2001.

[HS03] I. Horrocks, U. Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. Artificial Intelligence 160(1–2): 79–104, 2004. (`http://citeseer.ist.psu.edu/585553.html`).

[HSG04] U. Hustadt, R.A. Schmidt, L. Georgieva. A Survey of Decidable First-Order Fragments and Description Logics. Journal on Relational Methods in Computer Science, Vol. 1: 251–276, 2004.

[HST99] I. Horrocks, U. Sattler, S. Tobies. Practical reasoning for expressive description logics. In Proc. of LPAR'99. LNAI 1705: 161–180, Springer Verlag, 1999.

[IW94] M. Ito, G. E. Weddell. Implication problems for functional constraints on databases supporting complex objects. Journal of Computer and System Sciences 49(3): 726–768, 1994.

[JCG95] P. Jeavons, D. Cohen, M. Gyssens. A unifying framework for tractable constraints. Proceedings 1st International Conference on Constraint Programming (CP'95). LNCS 976: 276–291, Springer-Verlag, 1995.

[Kay79] M. Kay. Functional grammar. In: Proceedings of the Fifth Annual Meeting of the Berkeley Linguistic Society: 142–158, Berkeley 1979.

[KCV83] P. C. Kanellakis, S. S. Cosmadakis, M. Y. Vardi. Unary inclusion dependencies have polynomial time inference problems. In: Proc. of 15th ACM Symposium on Theory of Computing: 264–277, 1983.

[KLW95] M. Kiefer, G. Lausen, J. Wu. Logical foundations of object-oriented frame-based languages. Journal of the ACM 42(4): 741–843, 1995.

[KM82] R. M. Kaplan, J. T. Maxwell III. Lexical-Functional Grammar: A formal system for grammatical representation. In: J. Bresnan (Hrsg.). The mental representation of grammatical relations: 297–302, Budapest 1982.

[KR97] H.–J. Klein, J. Rasch. Functional Dependencies for Object Databases: Motivation and Axiomatization. Bericht Nr. 9706, Christian-Albrechts-Univerität Kiel, Institut für Informatik und Praktische Informatik, 1997.

[KT90] D. Kozen, J. Tiuryn. Logics of Programs. In: J. Van Leeuwen (Hrsg.): Handbook of Theoretical Computer Science. Formal Models and Semantics: 789–840. Elsevier Science Publishers (North-Holland), Amsterdam 1990.

[KTW00] V. L. Khizder, D. Toman, G. Weddell. On decidability and complexity of description logics with uniqueness constraints. Informal Proc. of International Workshop on Description Logics DL2000, 2000.

[KTW01] V. L. Khizder, D. Toman, G. Weddell. On decidability and complexity of description logics with uniqueness constraints. In: J. Van den Bussche and V. Vianu (Hrsg.): ICDT 2001, LNCS 1973: 54–67, Springer-Verlag, 2001.

[KW03] V. L. Khizder, G. E. Weddell. Reasoning about Uniqueness Constraints in Object Relational Databases. IEEE Transactions on Knowledge and Data Engineering 15(5): 1295–1306, 2003.

[LAHS02] C. Lutz, C. Areces, I. Horrocks, U. Sattler. Keys, nominals, and concrete domains. LTCS-Report 02-04, TU Dresden, 2002.

[LL99] M. Levene, G. Loizou. How to prevent interaction of functional and inclusion dependencies. Information Processing Letters (NL) 71(3–4): 115–125, 1999.

[LL01] M. Levene, G. Loizou. Guaranteeing no interaction between functional dependencies and tree-like inclusion dependencies. Theoretical Computer Science (NL) 254(1–2): 683–690, 2001.

[Mit83] J. C. Mitchell. The implication problem for functional and inclusion dependencies. Information and Control 56: 154–173, 1983.

[NB03] D. Nardi, R. J. Brachmann. An Introduction to Description Logics. In: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider. The Description Logic Handbook: Theory, Implementation and Applications. Chapter 1: 5–44, Cambridge University Press, 2003.

[Neb90] B. Nebel. Terminological reasoning is inherently intractable. Artificial Intelligence 43:235–249, 1990.

[Neb91] B. Nebel. Terminological cycles: Semantics and computational properties. In: J. Sowa (Hrsg.). Principles of Semantic Networks. Morgan-Kaufmann, San Mateo 1991.

[NeS90] B. Nebel, G. Smolka. Representation and reasoning with attributive descriptions. In: K.-H. Bläsius, U. Hedtstück, C.-R. Rollinger (Hrsg.). Sorts and types in artificial intelligence. Lecture Notes in Artificial Intelligence 418: 112–139, Springer-Verlag, 1990.

[NeS91] B. Nebel, G. Smolka. Attributive description formalisms ... and the rest of the world. In: O. Herzog, C. Rollinger (Hrsg.). *Textunderstanding in LILOG. Integrating Computational Linguistics and Artificial Intelligence*: 439–452, Springer-Verlag, 1991.

[Scd91] K. Schild. A correspondence theory for terminological logics: Preliminary report. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney 1991.

[Sch89] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In: H. J. Levesque, R. Reiter (Hrsg.), Principles of Knowledge Representation and Reasoning. Proceedings of the 1st International Conserence: 421–431, Toronto 1989.

[Sch01] K.-D. Schewe. Design theory for advanced datamodels. In: M. E. Orlowska, J. F. Roddick. Proceedings 12th Australasian Database Conference. ADC 2001. ix+176: 3-9, IEEE Comput. Soc, 2001.

[SCM02] U. Sattler, D. Calvanese, R. Molitor. Relationships with other formalisms. In: F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P. F. Patel-Schneider (Hrsg.): The Description Logic Handbook. Chapter 4: 142–183, Cambridge University Press, 2002.

[ST93] K.-D. Schewe, B. Thalheim. Fundamental concepts of object oriented databases. Acta Cybernetica 11(1-2): 49–83, 1993.

[Str81] R. S. Streett. Propositional dynamic logic of looping and converse. STOC: 375-383, Milwaukee 1981.

[Str82] R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. Information and Computation 54: 121–141, 1982.

[Tee94] G. Teege. Making the Difference: A Subtraction Operation for Description Logics. In: J. Doyle, E. Sandewall, P. Torasso (Hrsg.). Principles of Knowledge Representation and Reasoning: Proc. of the 4th International Conference (KR94): 540–550, Morgan Kaufmann, 1994.

[TH02] D. Tsarkov, I. Horrocks. Abox satisfiability reduced to terminological reasoning in expressive description languages. Proc. of the 9th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2002). Lecture Notes in Artificial Intelligence 2514: 435–449, Springer-Verlag, 2002.

[TH04] D. Tsarkov, I. Horrocks. Efficient Reasoning with Range and Domain Constraints. Proc. of the 2004 Description Logic Workshop (DL 2004): 41–50, 2004.

[Tob00] S. Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. In: Journal of Artificial Intelligence Research 12: 199–217, 2000.

[TW01] D. Toman, G. Weddell. On attributes, roles, and dependencies in description logics and the Ackermann case of the decision problem. In: Poc. of Description Logics 2001. CEUR-WS vol. 49: 76–85, 2001.

[TW03] D. Toman, G. Weddell. On Reasoning about structural equality in XML: A description logic approach. In: Proc. of ICDT 2003. LNCS 2572: 96–110, 2003.

[vBW94] M.F. van Bommel, G.E. Weddell. Reasoning about equations and functional dependencies on complex objects. IEEE Transactions on Knowledge and Data Engineering 6(3): 455–469, 1994.

[vEB97] P. van Emde Boas. The convenience of tilings. In: Complexity, logic, and recursion theory. Lecture Notes in pure and applied mathematics 157: 331–363, Marcel Decker Inc., 1997.

[VW86] M. Y. Vardi, P. Wolper. Automata-theoretic techniques for modal logics of programs. J. of Computer and System Sciences 32(2): 183–221, 1986.

[Wed92] G. Weddell. Reasoning about functional dependenciesgeneralized for semantic data models. ACM Transactions on Database Systems 17(1): 32–64, 1992.

[Weg93] I. Wegener. Theoretische Informatik: Eine algorithmenorientierte Einführung. Teubner-Verlag, Stuttgart 1993.

[Wes00] M. Wessel. Decidable and Undecidable Extensions of ALC with ALC with Composition-Based Role Inclusion Axioms. Mitteilung Nr. 301/01, Fachbereich Informatik der Universität Hamburg, 2000.

[WHB92] T.P. van der Weide, A. H. M. ter Hofstede, P. van Bommel. Uniquest: determining the semantics of complex uniqueness constraints. In: Computer Journal (UK) 35(2): 148–156, 1992.

[ZO97] X. Zhang, Z. M. Ozsoyoglu. Implication and referential constraints: A new formal reasoning. IEEE Transactions on Knowledge and Data Engineering 6: 894–910, 1997.