Direct Mapping
Konvertierung von Ontologien zu Logikprogrammen

Michael Hanselmann

Michael.Hanselmann@informatik.uni-ulm.de

Zusammenfassung Das Direct Mapping ist ein Verfahren, das die Konvertierung von beschreibungslogischen Axiomen in Logikprogrammregeln (und umgekehrt) ermöglicht. Es kann auf einer Teilmenge des Schnittes von Beschreibungslogik und Logikprogrammen angewendet werden. Die Mächtigkeit der so definierten Sprache entspricht etwa der von OWL Lite. Das Direct Mapping bietet die Möglichkeit, die Stärken der beiden derzeit bedeutensten Repräsentationsformalismen, den Beschreibungslogiken und den Logikprogrammen, zu verknüpfen - so auch im Kontext des Semantic Webs.

1 Einführung

Wissensmodellierung gewinnt in der heutigen Zeit zunehmend an Bedeutung - und dies in vielen Bereichen der Wissenschaft. Aktuelle Fragestellungen gibt es in der Medizin (Diagnoseunterstützung), in Informationssystemen (z.B. Textverstehen oder Übersetzung), in der strategischen Planung, in die Kontextwissen mit einfließen muss, oder beim Entwurf des Semantic Web, das weit komplexere Fragestellungen beantworten können soll, als die einfache Suche nach simplen Zeichenketten.

In aktuellen Wissensrepräsentationssystemen sind vor allem zwei Formalismen stark vertreten: die Beschreibungslogik, die nach derzeitigem Stand auch die Basisrepräsentationsform für Wissen im Semantic Web darstellen wird [4], und die Logikprogramme, die den meisten derzeit bedeutenden regelbasierten Systemen (wie beispielsweise Prolog) zu Grunde liegen [1].

Diese beiden Ansätze wurden jedoch lange Zeit nicht miteinander verknüpft, sondern weitgehend separat betrachtet. Für das Semantic Web, das beide Techniken nutzen wird - Regeln auf der einen[1], beschreibungslogische Ansätze auf der andern Seite - ist es wünschenswert, diese Brücke zu schlagen. Dies würde die Interaktionsmöglichkeiten von Anwendern mit informationsverarbeitenden Systemen und Systemen untereinander im Inter- oder Intranet deutlich erweitern [2].

Das von Benjamin N. Grosof et al. vorgeschlagene *Direct Mapping* [1] setzt genau hier an. Das Verfahren leistet eine Konvertierung von beschreibungslogischen Axiomen zu Logikprogramm-Regeln. Das Mapping kann allerdings nur

[1] es ist z.B. von Interesse, Regeln auf Ontologien aufzusetzen - vgl. [1], Kapitel 6

auf einer speziellen Teilmenge logischer Formeln angewendet werden, die als *Description Logic Programs* (DLP) bezeichnet wird. Die Zusammenhänge zwischen Beschreibungslogik, Logikprogrammen und DLP können Abbildung 1 entnommen werden.

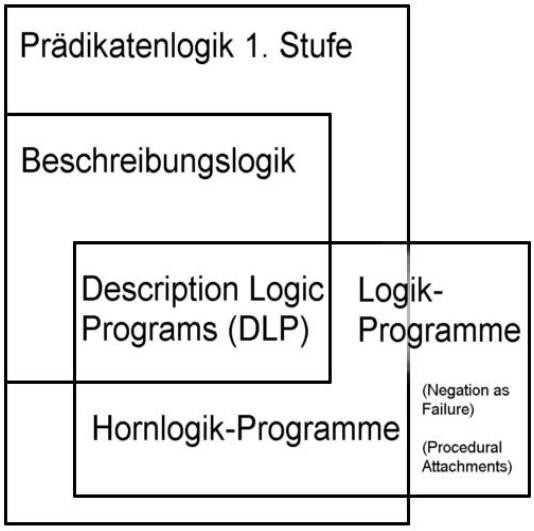

Abbildung 1. Schnittmenge von Beschreibungslogik und Logikprogrammen (vgl. [1],[2]). Die Beschreibungslogik ist eine Teilmenge der Prädikatenlogik 1. Stufe; ihren Schnitt mit der Menge der Logikprogramme bezeichnen Grosof et al. als DLP.

Das folgende Kapitel befasst sich näher mit der Beschreibungslogik und den Logikprogrammen. In Abschnitt 3 wird das Direct Mapping-Verfahren beschrieben, Kapitel 4 beleuchtet dessen Stärken und Schwächen und stellt ein alternatives Konvertierungsverfahren, das Meta Mapping, vor.

2 Grundlagen

2.1 Beschreibungslogik

Die Beschreibungslogik[2] wurde entwickelt, um einen Repräsentationsformalismus mit formaler Semantik, Trennung von objektbezogenem und konzeptionellen Wissen und leicht zugänglicher Syntax zu erschaffen, um ihn zur Formalisierung von semantischen Netzen und Frames einzusetzen. Dieser Ansatz führte zur Entstehung einer ganzen Logikfamilie, die aus den *structured inheritance networks* [8] hervorging.

[2] engl. Description Logic (DL)

Die Beschreibungslogik ist eine entscheidbare Teilmenge der Prädikatenlogik erster Stufe[3] und hat sich als ideale Basis für die Repräsentation und Schlußfolgerung über Ontologien herausgestellt. Als Ontologie bezeichnet man die Aufzählung aller für eine Wissensdomäne relevanten Konzepte und Relationen sowie deren hierarchische Anordnung. Desweiteren werden in ihr alle Eigenschaften konkreter Instanzen festgelegt.

Die Erkenntnis, dass Ontologien eine entscheidende Bedeutung für das Semantic Web einnehmen werden [5], führte schließlich zur Entwicklung der Ontologie-Sprache RDFS[4], der "Basis" des Semantic Web. Deren Ausdrucksmächtigkeit reichte jedoch nicht für alle Problemstellungen aus; so lassen sich in RDFS beispielsweise keine transitiven Relationszusammenhänge modellieren [5]. Aus diesem Grund entstanden weitere Sprachen auf Basis der Beschreibungslogik, darunter vor allem DAML+OIL und dessen Nachfolger OWL (Web Ontology Language), die Sprache, die vom W3C-Kommitee als Standard für die Repräsentation von Ontologien im World Wide Web vorgeschlagen wurde (vgl. [5]). DAML+OIL ist in ihrer Mächtigkeit mit der $\mathcal{SHOIQ}$(D) Beschreibungslogik vergleichbar [1].

Tabelle 1. Syntax der Beschreibungslogik [1],[4]

Beschreibungslogik	Prädikatenlogik 1. Stufe
$\top$	true
$\bot$	false
a:C	C(a)
⟨a,b⟩:P	P(a,b)
C$\sqsubseteq$D	$\forall$x.C(x) $\rightarrow$ D(x)
$P^+\sqsubseteq$P	$\forall$x,y,z.((P(x,y) $\wedge$ P(y,z)) $\rightarrow$ P(x,z)
$\top \sqsubseteq \leq 1$ P	$\forall$x,y,z.(P(x,y) $\wedge$ P(x,z)) $\rightarrow$ y=z
$C_1 \sqcap ... \sqcap C_n$	$C_1(x) \wedge ... \wedge C_n(x)$
$C_1 \sqcup ... \sqcup C_n$	$C_1(x) \vee ... \vee C_n(x)$
$\neg$C	$\neg$C(x)
P$\equiv Q^-$	$\forall$x,y.P(x,y) $\Leftrightarrow$ Q(y,x)
$\{a_1 \; ... \; a_n\}$	x=$a_1 \vee ... \vee$ x=a_2
$\exists$P.C	$\exists$y.(P(x,y) $\wedge$ C(y))
$\forall$P.C	$\forall$y.P(x,y) $\rightarrow$ C(y))
$\geq$n P.C	$\exists y_1 \; ... \; y_n . \bigwedge_{1 \leq i \leq n}$(P(x,$y_i$) $\wedge$ C(y_i)) $\wedge \bigwedge_{1 \leq i < n, i < j \leq n} y_i \neq y_j$
$\leq$(n-1) P.C	$\forall y_1 \; ... \; y_n . (\bigwedge_{1 < i < n}$(P(x,$y_i$) $\wedge$ C(y_i))) $\rightarrow$ ($\bigvee_{1 < i < n, i < j < n} y_i = y_j$)

C und D sind Klassen, P und Q Relationen, $\top$ ist das allgemeinste Konzept und eine Abkürzung für A$\sqcup\neg$A für eine (beliebige) Klasse A, $\bot$ ist das Symbol für das speziellste Konzept, P^- steht für das Inverse von P, P^+ steht für die transitive Hülle von P

[3] engl. First Order Logic (FOL)

[4] Resource Description Framework Schema

Syntax Die Grundbausteine der Beschreibungslogik bilden einfache Konzepte und binäre Relationen, mit denen die wichtigen Objekte der Anwendungsdomäne und deren Beziehungen untereinander modelliert werden. Diese werden dann mit verschiedenen Konstruktoren[5] zu komplexeren Ausdrücken kombiniert. Man unterscheidet dabei zwischen der TBox, in der allgemeine Konzepte definiert werden (konzeptionelles Wissen), und ABox, die eine Menge von individuellen Objekten bezüglich dieser Terminologie darstellt (objektbezogenes Wissen) [6].

Die Syntax der Beschreibungslogik kann Tabelle 1 entnommen werden, Klassenkonstruktoren, wie sie in DAML+OIL gebraucht werden, sind in Tabelle 2 aufgeführt.

Tabelle 2. Klassenkonstruktoren in DAML+OIL, vgl.[1]

Konstruktor ($\mathcal{SHOIQ}$ Axiom)	Syntax Beschreibungslogik	Beispiel
intersectionOf	$C_1 \sqcap ... \sqcap C_n$	Tier⊓Reptil
unionOf	$C_1 \sqcup ... \sqcup C_n$	Vögel⊔Reptilien
complementOf	¬C	¬Reptil
hasClass	∃P.C	∃lebtInAfrika.Hund
toClass	∀P.C	∀hatFlügel.Vogel
oneOf	$\{i_1 ... i_n\}$	{Chip, Chap}
hasValue	∃P.{i}	∃lebtIn.{Afrika}
minCardinalityQ	≥n P.C	≥2 hatBeine.Tier
maxCardinalityQ	≤n P.C	≤1001 hatBeine.Tier
cardinalityQ	= P.C	=4 hatBeine.Hund

2.2 Logikprogramme

Logikprogramme basieren auf der sogenannten Horn-Logik und verwenden Regeln der folgenden Gestalt

$$A \leftarrow B_1 \wedge ... \wedge B_m \wedge \sim B_{m+1} \wedge ... \wedge \sim B_n$$
$$mit\ n \geq 0\ und\ A\ und\ B_i\ Atome \quad (1)$$

Dabei bezeichnet man A als *Kopf* der Regel und $B_1 \wedge ... \wedge B_m \wedge \sim B_{m+1} \wedge ... \wedge \sim B_n$ als deren *Rumpf*. Klauseln mit leerem Rumpf nennt man *Fakten*. Der Implikationspfeil kann als "wenn" interpretiert werden. Die Notation $\sim$ steht für *Negation as failure*. In Logikprogrammen legt man die sogenannte *closed world assumption* zu Grunde; man nimmt also an, dass sämtliche relevanten Informationen in der Wissensbasis vorhanden sind. Alles, was auf Basis dieser Fakten

[5] wie zum Beispiel Durchschnitt, Vereinigung oder Komplement

[6] Das T von TBox steht für Terminologie, das A von ABox für Aussagen (engl. assertions)

nicht als *wahr* bewiesen werden kann, wird somit als *falsch* klassifiziert. Im Gegensatz dazu verwendet man bei Beschreibungslogiken die *open world assumption*. Fakten, die als *falsch* zu klassifizieren sind, müssen hier direkt angegeben werden.

Da Negation as failure in Beschreibungslogiken nicht dargestellt werden kann, beschränkt man sich beim in Kapitel 3 vorgestellten Konvertierungsverfahren, dem Direct Mapping, auf eine Teilmenge der Logikprogramme. Diese Teilmenge bezeichnet man mit dem Begriff *definite LP*[7].

Definite LP ist eng verwand mit Hornklauseln, sowohl syntaktisch als auch semantisch. Formeln, die in KNF vorliegen (also als Konjunktion von Disjunktionen von sogenannten Literalen, wobei ein Literal eine positive oder negative Atomformel ist), und die höchstens ein positives Literal enthalten, heißen Hornformeln. Die Disjunktionsglieder in einer KNF-Formel nennt man Klauseln. Hornklauseln, die genau ein positives Literal enthalten, werden als *definite Hornklauseln* bezeichnet. Die darin auftretenden Variablen sind allquantisiert, die Aussage

$$Hengst(x) \leftarrow Pferd(x) \wedge Männchen(x) \tag{2}$$

kann äquivalent auch als

$$\forall x.Hengst(x) \leftarrow Pferd(x) \wedge Männchen(x) \tag{3}$$

geschrieben werden.

def-LP und def-Horn Falls darüber hinaus kein Gleichheitsprädikat und keine Funktionssymbole[8] in den Hornklauseln auftreten (eine Sprache mit letzterer Eigenschaft ist *Datalog*), so spricht man von *def-LP*. Das prädikatenlogische Äquivalent hierzu ist *def-Horn*, eine Teilmenge der Prädikatenlogik erster Stufe, die definit und gleichheitsfrei ist, die Eigenschaft Datalog besitzt und aus Hornklauseln aufgebaut ist. Beide Mengen enthalten dieselben Fakten, Schlußfolgerungen in def-LP sind genauso gültig in def-Horn. In der umgekehrten Richtung gilt dies allerdings nur mit einer Einschränkung: Ableitungen in def-LP müssen immer die Form von Fakten haben, in def-Horn dagegen kann die Schlußfolgerung auch eine weitere Klausel sein. Die beiden Regeln

$$\begin{aligned} &AttraktiverMann(Peter) \leftarrow reich(Peter) \wedge gutaussehend(Peter) \\ &reich(Peter). \end{aligned} \tag{4}$$

lassen in def-Horn die Ableitung der Regel

$$AttraktiverMann(Peter) \leftarrow gutaussehend(Peter) \tag{5}$$

zu, in def-LP ist dies nicht möglich. Diese Einschränkung nimmt man jedoch häufig in Kauf, da def-LP eine wesentlich bessere Berechnungskomplexität als

[7] LP = Logic Program (Logikprogramm)
[8] 0-stellige Funktionen, also Konstanten, sind erlaubt

def-Horn aufweist[9], was in der Praxis meist wichtiger ist, als die beschriebene erweiterte Ableitbarkeit. Die Ähnlichkeit von def-LP und def-Horn nutzt man beim Direct Mapping aus.

2.3 Unterschiede in der Ausdrucksmächtigkeit

Sowohl die Beschreibungslogik als auch def-Horn sind in ihrer Ausdrucksmächtigkeit beschränkt. Dies hat zur Folge, dass Konvertierungsverfahren nur auf eine Teilmenge von Prädikatenlogik erster Stufe bzw. definiten Hornklauseln anwendbar sein können. Im folgenden werden diese Einschränkungen näher beleuchtet [vgl. 1,4].

Variablen in Regeln definiter Hornlogik sind grundsätzlich über die ganze Regel allquantisiert, Existenzquantoren dürfen nur im Rumpf auftreten. Die folgende Aussage in Beschreibungslogik, die fordert, dass zu einer Ehefrau ein Ehemann existieren muss, kann also nicht als definite Hornformel dargestellt werden[10]:

$$Ehefrau \sqsubseteq Frau \sqcap \exists verheiratetMit.Ehemann \tag{6}$$

Desweiteren ist der Gebrauch von Negation weder in Kopf noch Rumpf der Regel erlaubt. Die untenstehende Definition kann also ebensowenig formuliert werden:

$$\begin{aligned} &Person \sqsubseteq Mann \sqcup Frau \\ &Mann \sqsubseteq \neg Frau \end{aligned} \tag{7}$$

Bei Beschreibungslogiken ist der Gebrauch von Variablen und Quantoren eng festgelegt: die mit einem Quantor versehene Variable muss zwingend zusammen mit einer freien Variablen in einer Relation auftreten. Dies macht es unmöglich Klassen zu definieren, deren Instanzen mit einer anderen noch unbestimmten Instanz über zwei Relationen hinweg in Beziehung stehen. Der Begriff "Heimarbeiter", der ausdrücken soll, dass eine Person am selben Platz lebt und arbeitet, kann zwar mit einer Hornklausel, nicht aber beschreibungslogisch formuliert werden.

$$Heimarbeiter(x) \leftarrow arbeitet(x, y) \wedge lebt(x, z) \wedge Platz(y, w) \wedge Platz(z, w) \tag{8}$$

Instanzen der Klasse Heimarbeiter stehen in diesem Fall mittels der Relationen *arbeitet* und *lebt* mit dem noch unbestimmten Ort (y bzw. z) in Beziehung. Generell werden bei Beschreibungslogiken nur Konstanten und binäre Prädikate zugelassen, nicht aber mehrstellige Relationen. Darüber hinaus besteht der bereits erwähnte Unterschied in der Verwendung der closed- bzw. open-world-Annahme.

[9] polynominelle Berechnungszeit im Vergleich zu exponentieller Zeit

[10] die in [4] auf Seite 19 angegebene Umwandlung in die Regel *Ehefrau(X):-Frau(X),verheiratetMit(X,Y), Ehemann(Y).* ist nicht korrekt!

3 Direct Mapping

Das Direct Mapping kann in beide Richtungen angewandt werden (von Beschreibungslogik nach def-Horn oder umgekehrt), ohne dass sich die Semantik verändert. Meist wird es jedoch verwendet, um beschreibungslogische Axiome in def-Horn-Regeln zu konvertieren. Die Präferenz dieser Umwandlungsrichtung hat mehrere Gründe: zum einen können auf diese Weise die zahlreichen Schlußfolgerungsalgorithmen, die für Logikprogramme existieren, genutzt werden, zum anderen ist die Möglichkeit, Regeln auf Ontologien aufzusetzen, von großem Interesse für das Semantic Web (vgl. [1]).

Unter Berücksichtigung der im letzten Kapitel beschriebenen Einschränkungen werden nun die benötigten Konvertierungsregeln definiert, um beschreibungslogische Axiome in def-Horn-Regeln umzuwandeln. Im Folgenden seien C und D Klassen, P und Q dagegen Relationen. a und b seien Individuen.

3.1 A-Box-Axiome

Konzepte und Relationen werden in der Beschreibungslogik durch Axiome der Form

$$a : C \tag{9}$$

bzw.

$$\langle a, b \rangle : P \tag{10}$$

instanziiert. Diese können in def-Horn durch C(a) und P(a,b) ausgedrückt werden, wobei a und b Konstanten sind. Dabei bedeutet C(a), dass a eine Instanz der Klasse C ist, und P(a,b), dass Instanz a mit Instanz b über Relation P in Beziehung steht.

3.2 T-Box-Axiome

Unterklassen und Unterrelationen Die Inklusionsaxiome für Klassen und Relationen der Form

$$C \sqsubseteq D \tag{11}$$

(d.h. C ist eine Unterklasse von D) und

$$Q \sqsubseteq P \tag{12}$$

(d.h. Q ist eine Unterrelation von P) werden in

$$D(x) \leftarrow C(x) \tag{13}$$

und

$$P(x, y) \leftarrow Q(x, y) \tag{14}$$

konvertiert. Dieser direkten Umwandlung von Unterklassenbeziehungen in die zugehörigen Logikprogrammregeln verdankt das Direct Mapping seinen Namen, der zuerst in [2] erwähnt wurde.

Werte- und Funktionsbereich von Relationen Die Relationsrestriktionen für Werte- und Funktionsbereich werden durch die beschreibungslogische Konstrukte $\top \sqsubseteq \forall$P.C (Wertebereich von P ist C) bzw. $\top \sqsubseteq \forall\ P^{-}$.C (Domäne von P ist C) formuliert. Die Konvertierung verläuft dann nach diesem Schema:

$$\top \sqsubseteq \forall P.C \tag{15}$$

und

$$\top \sqsubseteq \forall P^{-}.C \tag{16}$$

werden zu

$$C(y) \leftarrow P(x, y) \tag{17}$$

und

$$C(y) \leftarrow P(y, x) \tag{18}$$

Äquivalenzen Der Äquivalenzoperator $\equiv$ stellt nichts anderes als eine symmetrische Inklusionsbeziehung dar und kann auch genau so behandelt werden. Aus

$$C \equiv D \tag{19}$$

bzw.

$$P \equiv Q \tag{20}$$

werden somit jeweils zwei Regeln, nämlich

$$\begin{aligned} D(x) &\leftarrow C(x) \\ C(x) &\leftarrow D(x) \end{aligned} \tag{21}$$

bzw.

$$\begin{aligned} Q(x, y) &\leftarrow P(x, y) \\ P(x, y) &\leftarrow Q(x, y) \end{aligned} \tag{22}$$

Inverse Relationen Sind zwei Relation invers zueinander, in beschreibungslogischer Notation

$$P \equiv Q^{-} \tag{23}$$

so lauten die zugehörigen Regeln in def-Horn

$$\begin{aligned} Q(y, x) &\leftarrow P(x, y) \\ P(x, y) &\leftarrow Q(y, x) \end{aligned} \tag{24}$$

Transitivität Transitive Relationen werden in der Beschreibungslogik durch

$$P^{+} \sqsubseteq P \tag{25}$$

beschrieben. Das Direct Mapping wandelt diese Definition in die Regel

$$P(x, z) \leftarrow P(x, y) \wedge P(y, z) \tag{26}$$

um.

3.3 Konstruktoren

Wie bereits erwähnt können in der Beschreibungslogik Klassen und Relationen mit Hilfe von verschiedenen Konstruktoren zu komplexeren Ausdrücken kombiniert werden. Ob ein Ausdruck überhaupt nach def-Horn konvertiert werden kann hängt davon ab, auf welcher Seite der Regel diese Konstruktoren auftreten.

Konjunktion Die Konjunktion von Klassen kann problemlos nach def-Horn umgewandelt werden, unabhängig davon, ob die Konjunktion $\sqcap$ links oder rechts des Unterklassenoperators $\sqsubseteq$ steht:

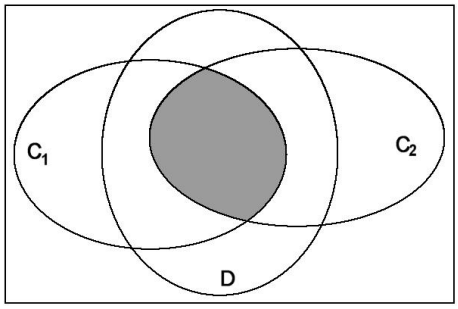

Abbildung 2. $C_1 \sqcap C_2 \sqsubseteq D$

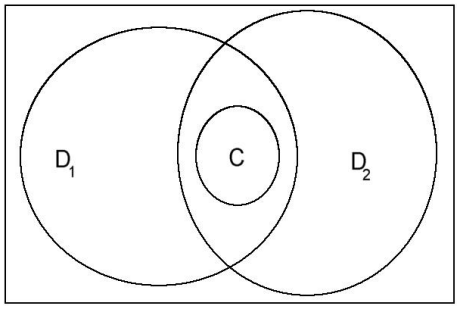

Abbildung 3. $C \sqsubseteq D_1 \sqcap D_2$

$$C_1 \sqcap C_2 \sqsubseteq D \tag{27}$$

wird zu

$$D(x) \leftarrow C_1(x) \wedge C_2(x) \tag{28}$$

und

$$C \sqsubseteq D_1 \sqcap D_2 \tag{29}$$

wird zu

$$\begin{aligned} D_1(x) &\leftarrow C(x) \\ D_2(x) &\leftarrow C(x) \end{aligned} \tag{30}$$

Die Abbildungen 2 und 3 veranschaulichen den Zusammenhang der gegebenen Mengen graphisch.

Disjunktion Problematischer gestaltet sich die Handhabung von Disjunktionen. Tritt eine solche Verknüpfung von Klassen auf der rechten Seite des Unterklassenoperators auf, so muss das zugehörige Axiom zu einer Regel konvertiert werden, in deren Kopf ebenfalls eine Disjunktion steht. Eine solche Regel ist in def-Horn jedoch nicht zugelassen. Folglich muss dafür gesorgt werden, dass Axiome dieser Bauart nicht in Wissensbasen auftreten, auf denen das Direct Mapping durchgeführt werden soll. Disjunktionen auf der linken Regelseite machen dagegen keine Probleme. Das beschreibungslogische Axiom

$$C_1 \sqcup C_2 \sqsubseteq D \tag{31}$$

wird auf die Regeln

$$\begin{aligned} D(x) &\leftarrow C_1(x) \\ D(x) &\leftarrow C_2(x) \end{aligned} \tag{32}$$

abgebildet. Abbildung 4 zeigt, dass die Mengen C_1 und C_2 beide innerhalb der Menge D liegen. Ihr Schnitt kann leer sein oder auch Elemente enthalten.

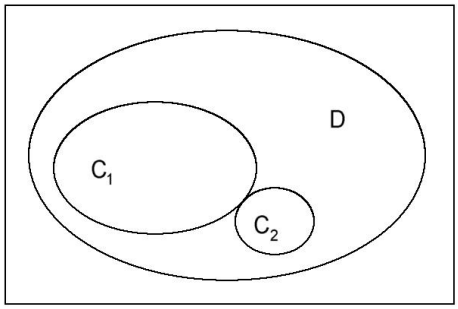

Abbildung 4. $C_1 \sqcup C_2 \sqsubseteq D$

Aus Abbildung 5 ist ersichtlich, dass aus C $\sqsubseteq$ $D_1 \sqcup D_2$ nicht abgeleitet werden kann, dass C vom Typ D_1 oder vom Typ D_2 ist.

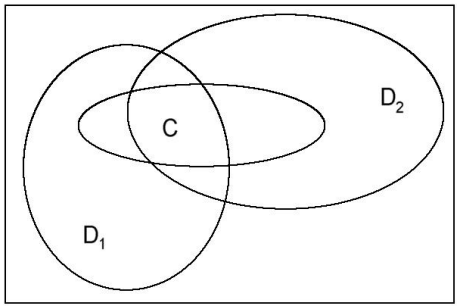

Abbildung 5. C $\sqsubseteq$ $D_1 \sqcup D_2$

Allquantor Ein ähnliches Problem tritt beim Mapping von Formeln mit dem Allquantor, der in der Form ∀P.C verwendet werden kann, auf. Zwar können Ausdrücke der Art

$$C \sqsubseteq \forall P.D \tag{33}$$

in def-Horn durch

$$\begin{array}{l}(D(y) \leftarrow P(x,y)) \leftarrow C(x) \; oder \; äquivalent \\ D(y) \leftarrow C(x) \wedge P(x,y)\end{array} \tag{34}$$

dargestellt werden, nicht aber Restriktionen, bei denen der Allquantor auf der linken Seite von $\sqsubseteq$ verwendet wird. Grund dafür ist die Tatsache, dass Negationen in def-Horn-Klauseln nicht gestattet sind[11]. Diese können jedoch beim Mapping solcher Ausdrücke entstehen. Das beschreibungslogische Axiom

$$\forall P.C \sqsubseteq D \tag{35}$$

müsste in die Regel

$$\begin{array}{l}D(x) \leftarrow (C(y) \leftarrow P(x,y)) \; oder \; äquivalent \\ D(x) \leftarrow \neg(\neg C(y) \wedge P(x,y))\end{array} \tag{36}$$

konvertiert werden. Dies hat aber die Negation von P(x,y) zur Folge.

[11] dies gilt nur für Axiome, die auch durch Umformungen nicht negationsfrei dargestellt werden können

Existenzquantor Umgekehrt verhält es sich mit dem Existenzquantor, der in der Form $\exists P.C$ auftreten kann. Hier können Axiome der Art

$$\exists P.C \sqsubseteq D \tag{37}$$

in

$$D(x) \leftarrow P(x, y) \wedge C(y) \tag{38}$$

gewandelt werden, nicht aber solche, bei denen der Existenzquantor rechts eingesetzt wird. Die zum Axiom

$$C \sqsubseteq \exists P.D \tag{39}$$

gehörende Regel müsste wie folgt lauten:

$$P(x, y) \wedge D(y) \leftarrow C(x) \tag{40}$$

Problematisch hierbei ist, dass in diesem Fall der Kopf der (durch Konvertierung erzeugten) Regel eine Konjunktion mit einer existentiell quantifizierten Variablen (y) besitzt. Dies darf in def-Horn nicht sein.

Negation und Kardinalitätseinschränkungen Ausdrücke mit Negationen lassen sich nicht nach def-Horn konvertieren, da Negation weder im Kopf noch im Rumpf der def-Horn-Klausel erlaubt ist. Ebensowenig können Kardinalitätseinschränkungen umgewandelt werden[12], da diese wie aus Tabelle 1 ersichtlich ist auf Variablengleich- oder -ungleichheit zurückgeführt werden können. def-Horn schließt die Verwendung des Gleichheitsprädikats jedoch aus.

3.4 Konvertierungsfunktion

Die beschriebenen Regeln des Mappings lassen sich als rekursive Konvertierungsfunktion $\mathcal{T}$ darstellen. Um der dargestellten Problematik der von der Stellung der Konstruktoren abhängigen Konvertierbarkeit Sorge zu tragen, unterscheidet man zwischen zwei Klassen von beschreibungslogischen Sprachen: $\mathcal{L}_h$ und $\mathcal{L}_b$. Dabei ist $\mathcal{L}_h$ diejenige Sprache, von der aus in den Kopf der def-Horn-Regeln konvertiert werden kann, und $\mathcal{L}_b$ deren Äquivalent für die Umwandlung in den Rumpf dieser Regeln. Die Mappingfunktion konvertiert dabei die Unterklassenbeziehung der Klassen C (aus $\mathcal{L}_b$) und D (aus $\mathcal{L}_h$) in eine Logikprogrammregel der Form A $\leftarrow$ B:

$$\mathcal{T}(C \sqsubseteq D) \longrightarrow Th(D, y) \leftarrow Tb(C, y) \tag{41}$$

An dieser Stelle wird davon ausgegangen, dass alle Unterklassenbeziehungen (Konzepte und Relationen) in der Form A$\sqsubseteq$B vorliegen. Um dies zu erreichen können Axiome mit mehr als einem Unterklasseoperator paarweise in neue

[12] diese Aussagen gelten natürlich nur für Axiome, die auch durch Umformung nicht von diesen Konstruktoren "befreit" werden können

Axiome aufgeteilt werden. So wird beispielsweise A⊑B⊑C in die beiden Axiome A⊑B und B⊑C "zerlegt".

Th(D,y) und Tb(C,y) werden dann rekursiv weiterverarbeitet [1],[4]:

$$\begin{aligned}
&Th(A,x) \longrightarrow A(x)\\
&Th((C \sqcap D),x) \longrightarrow Th(C,x) \wedge Th(D,x)\\
&Th((\forall R.C),x) \longrightarrow Th(C,y) \leftarrow R(x,y)\\
&Tb(A,x) \longrightarrow A(x)\\
&Tb((C \sqcap D),x) \longrightarrow Tb(C,x) \wedge Tb(D,x)\\
&Tb((C \sqcup D),x) \longrightarrow Tb(C,x) \vee Tb(D,x)\\
&Tb((\exists R.C),x) \longrightarrow R(x,y) \wedge Tb(C,y)
\end{aligned} \tag{42}$$

Diese Auflistung muss noch um folgende Regeln ergänzt werden

$$\mathcal{T}(C \equiv D) \longrightarrow \mathcal{T}(C \sqsubseteq D)\ und\ \mathcal{T}(D \sqsubseteq C)$$

$$\begin{aligned}
&\mathcal{T}(\top \sqsubseteq \forall P.D) \longrightarrow Th(D,y) \leftarrow P(x,y)\\
&\mathcal{T}(\top \sqsubseteq \forall P^{-}.D) \longrightarrow Th(D,x) \leftarrow P(x,y)\\
&\mathcal{T}(a : D) \longrightarrow Th(D,a)\\
&\mathcal{T}(\langle a,b\rangle : P) \longrightarrow P(a,b)
\end{aligned}$$

$$\begin{aligned}
&\mathcal{T}(P \sqsubseteq Q) \longrightarrow Q(x,y) \leftarrow P(x,y)\\
&\mathcal{T}(P \equiv Q) \longrightarrow Q(x,y) \leftarrow P(x,y)\ und\ P(x,y) \leftarrow Q(x,y)\\
&\mathcal{T}(P \equiv Q^{-}) \longrightarrow Q(x,y) \leftarrow P(y,x)\ und\ P(y,x) \leftarrow Q(x,y)\\
&\mathcal{T}(P^{+} \sqsubseteq P) \longrightarrow P(x,z) \leftarrow P(x,y) \wedge P(y,z)
\end{aligned} \tag{43}$$

die die Konvertierung von Instanzen, Relationsrestriktionen (Werte- und Funktionsbereich), Äquivalenzen und Relationsbeziehungen abdecken.

3.5 DHL und DLP

Das Direct Mapping kann aufgrund der beschriebenen Einschränkungen also nur auf einer sehr kleinen Teilmenge des Schnittes von Beschreibungslogik und Logikprogrammen angewendet werden. Diese Menge bezeichnet man als *Description Horn Logic* oder *DHL*. Ihre Ausdrucksmächtigkeit ist mit der von OWL Lite, der ausdrucksschwächsten Variante von OWL, vergleichbar, allerdings um einige wenige Konstrukte aus OWL erweitert.[13].

Das zu den beim Mapping entstehenden def-Horn-Regeln gehörende Logikprogramm nennen Grosof et al. ein *Description Logic Program* oder *DLP*.

3.6 Beispiel

In Tabelle 3 wird ein konkretes Beispiel betrachtet, in dem eine in Beschreibungslogik gegebene Wissenbasis in Logikprogramm-Regeln umgeformt wird. Die Wissensbasis ist so gewählt, dass sie in vollem Umfang umgewandelt werden kann.

[13] Unterkonzeptbeziehungen mit Disjunktionen und den in Kapitel 3.3 beschriebenen Existenzquantoren auf der linken Seite des ⊑ - Operators, siehe dazu [2], Kapitel 2.2

Tabelle 3. Beispiel für die Anwendung des Direct Mappings, Konvertierung beschreibungslogischer Axiome in Logikprogrammregeln

Nr.	Beschreibungslogik	Logikprogrammregel(n)
T1	Weibchen $\sqsubseteq$ Lebewesen	Lebewesen(X):-Weibchen(X).
T2	Männchen $\sqsubseteq$ Lebewesen	Lebewesen(X):-Männchen(X).
T3	Tiger $\sqsubseteq$ Tier	Tier(X):-Tiger(X).
T4	Hund $\sqsubseteq$ Tier	Tier(X):-Hund(X).
T5	Rüde $\equiv$ Männchen$\sqcap$Hund	Männchen(X):-Rüde(X). Hund(X):-Rüde(X). Rüde(X):-Männchen(X), Hund(X).
T6	Dschungel$\sqcup$Wüste $\sqsubseteq$ Wildnis	Wildnis(X):-Dschungel(X). Wildnis(X):-Wüste(X).
T7	Haustier $\sqsubseteq$ Tier$\sqcap\forall$lebtIn.Haus	Tier(X):-Haustier(X). Haus(Y):-Haustier(X), lebtIn(X,Y).
T8	hatFreund$^-$ $\sqsubseteq$ hatFreund	hatFreund(X,Y):-hatFreund(Y,X).
T9	$\exists$lebtIn.{dschungel} $\sqsubseteq$ Dschungelbewohner	Dschungelbewohner(X):-lebtIn(X,dschungel).
T10	istGrößer$^+$ $\sqsubseteq$ istGrößer	istGrößer(X,Z):-istGrößer(X,Y), istGrößer(Y,Z).
A1	Rüde$\sqcap$Haustier(pluto)	I_1(pluto). Rüde(X):-I_1(X). Haustier(X):-I_1(X).
A2	Hund(rantanplan)	I_2(rantanplan). Hund(X):-I_2(X).
A3	Tiger(shirkhan)	I_3(shirkhan). Tiger(X):-I_3(X).
A4	Haus(donalds-hütte)	I_4(donalds-hütte). Haus(X):-I_4(X).
A5	lebtIn(pluto, donalds-hütte)	lebtIn(X,Y):-I_1(X), I_4(Y).
A6	hatFreund(pluto, rantanplan)	hatFreund(X,Y):-I_1(X), I_2(Y).
A7	istGrößer(shirkhan, rantanplan)	istGrößer(X,Y):-I_3(X), I_2(Y).
A8	istGrößer(rantanplan, pluto)	istGrößer(X,Y):-I_2(X), I_1(Y).

Beispielanfrage Erweitert man diese Axiome um die Definitionen aus Tabelle 4, so kann aus den gegebenen Fakten die Schlußfolgerung gezogen werden, dass idefix ein Rüde ist: A10 liefert zunächst die Information, dass idefix ein Männchen ist. Da er in einer Hundehütte wohnt (A9 und A11) kann abgeleitet werden, dass idefix auch ein Hund ist, denn Hundehütten können aufgrund von T11 nur von Hunden bewohnt werden. Aus T5 folgt dann, dass idefix ein Rüde ist.

Tabelle 4. Zusätzliche Definitionen

Nr.	Beschreibungslogik	Logikprogrammregel(n)
T11	Hundehütte $\sqsubseteq$ Haus$\sqcap\forall$bewohntVon.Hund	Haus(X):-Hundehütte(X). Hund(Y):-Hundehütte(X), bewohntVon(X,Y).
A9	Hundehütte(idefix-hütte)	I_5(idefix-hütte). Hundehütte(X):-I_5(X).
A10	Männchen(idefix)	I_6(idefix). Männchen(X):-I_6(X).
A11	bewohntVon(idefix-hütte, idefix)	bewohntVon(X,Y):-I_5(X), I_6(Y).

4 Bewertung

4.1 Stärken und Schwächen von Direct Mapping

Das Direct Mapping ermöglicht die Konvertierung von beschreibungslogischem Wissen in Logikprogramme und umgekehrt und eröffnet so die Möglichkeit, Schlußfolgerungen auf Ontologien auszuführen und Regeln auf ihnen aufzusetzen. Letzterer Punkt ist besonders für das Semantic Web von Interesse. Für die Schlußfolgerungen auf diesen Ontologien kann man auf zahlreiche und bewährte Programme, Theorembeweiser und Algorithmen, die bereits bei Logikprogrammen eingesetzt werden, zurückgreifen. Analog kann das Mapping auch dazu verwendet werden, um Logikprogrammregeln mit beschreibungslogischen Werkzeugen zu verarbeiten. Diesen Vorzügen stehen allerdings einige Probleme gegenüber.

Wie beschrieben liegt eine große Schwäche des Direct Mappings in der stark limitierten Ausdrucksmächtigkeit von DLP. Aufgrund der engen Restriktionen für den Gebrauch der verschiedenen Konstruktoren ($\sqcup$, $\forall$, $\exists$) bzw. deren "Verbot" (Kardinalitätsoperatoren, Negation) können häufig Informationen aus der Wissensbasis nicht oder nur teilweise modelliert werden.

Dies ist jedoch nicht die einzige Schwäche dieses Ansatzes. Da die Bezeichnungen von Klassen und Relationen innerhalb des Logikprogramms nicht über Prädikate zugreifbar sind, können einige übliche Anfragen nur über aufwendige Hilfskonstrukte beantwortet werden. Die Anfrage "Von welchen Klassen ist

Objekt X eine Instanz?" kann nur bearbeitet werden, indem für jede Klasse Y_i eine Anfrage der Art "Ist Objekt X eine Instanz von Klasse Y_i?" generiert und berechnet wird. Abgesehen von der Ineffizienz dieser Variante ist dafür die Kenntnis aller Klassenbezeichnungen notwendig. Besser wäre es, die Klassen und Relationen zu Argumenten von sogenannten "Meta-Prädikaten" und damit für Logikprogramme zugreifbar zu machen. Ein weiteres Problem besteht darin, dass die Anzahl der entstehenden Regeln bei der Konvertierung linear mit der Anzahl von Klassen- und Relationsdefinitionen wächst und nicht konstant bleibt (vgl. [2]).

Zudem gestaltet sich die Anfrageoptimierung äußerst kompliziert, da die Bezeichnungen und die Strukturen der betroffenen Regeln von Wissensbasis zu Wissensbasis variieren. Das in [2] vorgestellte *Meta Mapping* löst einige der angesprochenen Probleme besser.

4.2 Alternative: Das Meta Mapping

Wie beschrieben kann das Direct Mapping nicht alle Anfragen, die normalerweise an beschreibungslogische Systeme gerichtet werden, zufriedenstellend beantworten. Typische Fragestellungen können wie folgt lauten:

- ist die gegebene Instanz a vom Typ C? (A1)
- liste alle Instanzen des Typs C auf! (A2)
- von welchen Klassen ist das gegebene Objekt a eine Instanz? (A3)
- bestimme, ob zwei gegebene Klassen C und D in einer Unterklassenrelation stehen! (A4)
- liste alle Ober- bzw. Unterklassen einer Klasse C auf! (A5)
- ist Klasse C bezüglich der Wissensbasis erfüllbar? (A6)
- gib die speziellste Klasse zu einer Klasse C aus, d.h. die Unterklasse von C, die keine weiter Unterklasse besitzt! (A7)[14]

(A1) und (A2) können mittels des Direct Mappings problemlos bearbeitet werden, die Anfragen (A3) bis (A7) erfordern jedoch bereits ein aufwendigeres Vorgehen. Auf die Problematik bei (A3) wurde bereits eingegangen, um (A4) zu beantworten muss zunächst eine Instanz i der Klasse C erzeugt werden und dann eine neue Anfrage formuliert werden, die überprüft, ob i auch eine Instanz von Klasse D ist. Ähnliche Schwierigkeiten treten bei der Berechnung von (A4) oder (A7) auf.

Durch die Einführung von Meta-Prädikaten erlaubt es das Meta Mapping viele Anfragen geschickter zu stellen und zu bearbeiten. Dazu werden Instanzen nicht durch C(a) definiert, sondern über ein spezielles Prädikat *type*, das in der Form von type("a","C") verwendet wird. Analog dazu definiert man Relationen mit Hilfe des Prädikats *propInst*: propInst("P","a","b") statt P(a,b). Unterklassenbeziehungen werden durch isSub("Mann","Mensch") beschrieben, für die Konstruktoren gibt es ähnliche Regeln.[15] Damit sind dem Programm

[14] im Englischen wird diese als *most specific instantiator* (msi) bezeichnet

[15] eine komplette Auflistung der Regeln findet sich unter http://www.informatik.uni-ulm.de/ki/Liebig/MM-ruleset.txt

sämtliche Klassen- und Relationsnamen bekannt und können folglich auch erfragt werden:

Die Anfrage "?type("i",C)." liefert alle Klassen, von denen i eine Instanz ist.
Die Anfrage "?isSub(X,"C")." listet alle Unterklassen von Klasse X auf.

In beiden Fällen wäre eine Beantwortung durch das Direct Mapping nur durch ineffiziente Methoden möglich (vgl. oben).

Eine weitere Verbesserung ist, dass die Anzahl der beim Meta Mapping entstehenden Regeln unabhängig von der Größe der Ontologie sind. Daraus resultiert ein deutlicher Geschwindigkeitsvorsprung gegenüber dem Direct Mapping bei der Bearbeitung großer Wissensbasen [2]. Weiterhin ungelöst bleibt das Problem der geringen Ausdrucksmächtigkeit von DHL.

5 Zusammenfassung

Das Direct Mapping beschreibt einen Weg, wie beschreibungslogische Axiome in Logikprogrammregeln (und umgekehrt) konvertiert werden können. Diese (äquivalente) Umwandlung kann jedoch nur auf einer Teilmenge des Schnittes von Beschreibungslogik und Logikprogrammen durchgeführt werden, deren Ausdrucksmächtigkeit etwa der von OWL Lite entspricht. Dies stellt einen deutlichen Nachteil dieses Ansatzes dar. Der Hauptvorteil dieses Verfahrens ist es, dass es die Nutzung von Logikprogrammalgorithmen für Ontologien und beschreibungslogische Bearbeitung von Logikprogrammregeln erlaubt. Es ermöglicht damit die Nutzung zahlreicher effizienter und ausgereifter Algorithmen und Verfahren für Aussagen in beiden Logiken.

Das Meta Mapping stellt eine interessante und mächtigere Alternative zum Direct Mapping dar, welche die hier vorgestellten Verfahren und Überlegungen sinnvoll ergänzt.

Literatur

1. Benjamin N. Grosof, Ian Horrocks, Raphael Volz and Stefan Decker. Description Logic Programs: Combining Programs with Description Logic. In *Proceedings of the 12th International World Wide Web Conference,* Budapest, Hungary, May 2003.
2. Timo Weithöhner, Thorsten Liebig and Günther Specht. Storing and Querying Ontologies in Logic Databases. In *Proceedings of the Workshop "Semantic Web and Databases" at the VLDB 2003,* Berlin, Germany, September 2003.
3. Raphael Volz. Web Ontology Reasoning with Logic Databases. *Dissertationsarbeit,* Universität Karlsruhe, Februar 2004.
4. Ian Horrocks, Raphael Volz, Stefan Decker, Benjamin Grosof and Boris Motik. Rule Language. Wonder Web: Ontology Infrastructure for the Semantic Web. Juni 2003. http://wonderweb.semanticweb.org/deliverables/documents/D2.pdf
5. Ian Horrocks and Peter F. Patel-Schneider. Three Theses of Representation in the Semantic Web. In *Proceedings of the 12th International World Wide Web Conference,* Budapest, Hungary, May 2003.

6. Daniele Nardi and Ronald J. Brachman. Introduction to Description Logic. In *The Description Logic Handbook*, chapter 1, pages 1-39. Franz Baader and Diego Calvanese and Deborah L. McGuiness and Daniele Nardi and Peter F. Patel-Schneider, Cambridge, 2003.
7. Benjamin Grosof. Logic Programs as basic representation. In *Business Rules for E-Commerce: Interoperability and Conflict Handling*, Watson Research Center, Hawthorne, NY, USA, September 1999.
http://www.research.ibm.com/rules/paps/html/talk_proj_ovw_html
8. R.J. Brachman. On the Epistemological Status of Semantic Networks.. In *N.V. Findler (Ed.), Associative Networks: Representation and Use of Knowledge by Computers*, pp. 3-50. Academic Press, New York, 1979.