

# **Formale Betrachtung von Anfragen auf RDF Datenbanken**

## **Seminararbeit**

im Fachbereich Biologie und Informatik  
an der Johann Wolfgang Goethe Universität Frankfurt am Main

bei Herrn Prof. Dott. Ing. Zicari

betreut von Dipl. Math. Karsten Tolle

verfasst von  
Bartholomäus Ende

## Inhaltsverzeichnis

<b>1. Kurzfassung</b> .....	2
<b>2. Einleitung</b> .....	2
<b>3. Resource Description Framework</b> .....	3
3.1. Tripelschreibweise .....	3
3.2. Graphdarstellung .....	4
3.3. XML/RDF .....	4
<b>4. Grundlegende Definitionen</b> .....	5
4.1. RDF Graphen .....	5
4.2. Pseudographen .....	6
4.3. Interpretation von RDF Graphen .....	6
4.4. Minimale Repräsentation .....	10
<b>5. Anfragen auf RDF Datenbanken</b> .....	12
5.1. Abfragesprache .....	12
5.2. Antworten auf Anfragen .....	13
5.3. Reification .....	15
<b>6. Komplexitätsbetrachtungen</b> .....	16
6.1. Berechnung von Antworten .....	16
6.2. Minimierung von Abfragen .....	17
6.3. Eliminierung von Redundanzen .....	19
<b>7. Schlussfolgerungen und RAL</b> .....	20
<b>8. Literaturverzeichnis</b> .....	21

## 1. Kurzfassung

Dieses Dokument befasst sich mit der formalen Analyse von Anfragen auf RDF-Datenbanken. Zu diesem Zweck wird zunächst eine kurze Einführung in das Resource Description Framework (RDF) gegeben.

Für die folgende formale Betrachtung der Abfrageverarbeitung werden sodann eine einfache Abfragesprache sowie zwei mögliche Definitionen von Antworten eingeführt und analysiert. Des Weiteren werden Aspekte wie die statische Optimierung von Abfragen, die Entfernung von Redundanzen aus Antworten sowie die Besonderheiten, die sich durch blank Nodes und Reification ergeben, behandelt.

Den Abschluss bildet eine Motivation von RAL, einer RDF-Algebra, die zukünftig nicht nur die Definition sondern auch den Vergleich von unterschiedlichen Abfragesprachen ermöglichen könnte. Zudem bietet sie auch das Potential, die bei relationalen Datenbanken üblichen algebraischen Optimierungen auch auf RDF Anfragen anzuwenden.

## 2. Einleitung

Die rasante Entwicklung von Informationstechnologien führt zu immer neuen Herausforderungen für die Verwaltung von Daten. Hierbei generiert speziell die Realisierung einer effizienten Nutzung der Inhalte, die durch das WWW bereitgestellt werden, neue Herausforderungen für den Zugriff auf die gebotene Flut von multimedialen Informationen.

Eine davon ist die Definition einer Repräsentationsform für Inhalte, die nicht nur für Menschen sondern auch für Maschinen verständlich ist. Erst mit dieser kann das WWW sein gesamtes Potential als ein Medium entfalten, das es seinen Benutzern, zu denen auch automatisierte Prozesse gehören, ermöglicht mit Informationen in einer wohldefinierten Art und Weise umzugehen.

Diese Vision, die weit über die bloße Darstellung von Inhalten reicht, wurde von dem W3C Konsortium unter dem Begriff des *Semantic Web* definiert und basiert zum großen Teil auf dem *Resource Description Framework (RDF)*, das die Automatisierung, Integration und somit die Wiederverwendung von Informationen über eine Vielzahl von Anwendungen ermöglicht, in dem es eine standardisierte Form für Metadaten ermöglicht.

Unter *Metadaten* wird hier nichts anderes als eine Beschreibung und Charakterisierung der tatsächlichen Inhalte verstanden, d.h. Daten über Daten, die das Ziel verfolgen, die Semantik von Aussagen für Maschinen verständlich zu machen. Dabei ermöglichen sie bei Uneindeutigkeiten der Sprache die beabsichtigte Bedeutung von Aussagen zu identifizieren

Zu diesen Problemfällen gehören zum Beispiel Homonyme, d.h. mehrfache Bedeutungen eines Wortes (z.B. Maus), Synonyme, d.h. mehrere Wörter mit äquivalenten Bedeutungen (z.B. Computer und Rechner) oder Mehrdeutigkeiten, die erst aus dem Kontext aufgeklärt werden können. Allen drei Fällen ist gemein, dass es bei Suchanfragen, die sich lediglich auf die orthographische Übereinstimmung von Ausdrücken beschränken, zu einer Einschränkung der Ergebnismenge kommt. Durch den Einsatz von Metadaten kann dieses Problem größtenteils vermieden und somit die Qualität der Suchergebnisse deutlich verbessert werden.

Die Verwendung von Metadaten ist aber nicht nur auf die Recherche von Informationen beschränkt. Es ist auch möglich weiterreichende Informationen wie beispielsweise Urheberrechte, Vertragsbedingungen oder die Informationsformate selbst so darzustellen, dass sie für Maschinen verständlich werden um, so Tätigkeiten, die derzeit noch manuell vorgenommen werden müssen, automatisiert zu können.

Nachdem die Möglichkeiten des RDF kurz umrissen wurden, folgt im nächsten Abschnitt eine grundlegende Einführung in letzteres.

### 3. Resource Description Framework

*RDF* basiert auf einfachen Aussagen der Form „Subjekt, Prädikat, Objekt“, die jeweils einer Ressource eine Eigenschaft (Property) zuordnen, wobei das Prädikat die Art dieser Eigenschaft und das Objekt die Ausprägung definiert. Als *Ressourcen* werden alle Objekte angesehen, denen man für eine eindeutige Identifizierung einen URI (Uniform Resource Identifier) zuweisen kann. Da die Vergaben von eindeutigen URIs nicht nur auf digitale Daten, die über das WWW abrufbar sind, beschränkt ist, können somit auch Gegenstände wie ein Haus oder ein Baum beschrieben werden.

Des Weiteren existieren zusätzlich noch *anonyme Ressourcen*, so genannte blank Nodes, denen keine explizite URI zugewiesen wird, die jedoch innerhalb einer RDF Beschreibung bzw. Applikation eine eindeutige URI durch den RDF Prozessor zugewiesen bekommen, damit ihre Verwendung nicht nur auf ein einziges Statement beschränkt bleibt. Die dritte und letzte Art von Elementen innerhalb des RDFs bilden die *Literale*, die Zeichenketten entsprechen, welche zusätzlich für die Kennzeichnung des kodierten Datentyps eine Typisierung aufweisen können.

Bei der Verwendung dieser Elemente in RDF Statements gilt die Regel, dass Subjekte einer Ressource bzw. einem blank Node, Prädikate stets einem URI und Objekte einer beliebigen der drei möglichen Elementarten entsprechen. Aus dieser Definition folgt, dass Prädikate selbst Ressourcen sind und somit ihre Bedeutung mittels entsprechender Vokabulare klar definiert werden kann.

Hierfür wurde das RDF Schema (RDFS) eingeführt. Es ist selbst ein Vokabular, das die Möglichkeit bietet eigene Vokabulare, bestehend aus speziellen Klassen von Ressourcen sowie Properties, zu definieren und damit Anwendungen, die für dieses so geschaffene Vokabular zugeschnitten sind, ein tieferes Verständnis der Aussagen zu verschaffen.

Für eine einfache Definition sowie Wiederverwendung der eindeutigen Identifier einer jeden Ressource bedient sich das RDF dem Konzept der *Namensräume* (Namespaces). Dies sind Dateien, die eine Menge von Ressourcen mittels eines Schemas beschreiben. Die Verwendung der Ressourcen wird dadurch realisiert, dass der zu verwendende URI der Konkatenation des Unique Resource Locator (URL) des Namespaces, einer Raute als Trennzeichen und der innerhalb des Namespaces verwendeten ID als Suffix entspricht. Zudem werden für eine kürzere Schreibweise qualified Names (QName) verwendet, die einer Substitution der URL des Namespaces samt Trennraute entsprechen. Die folgende Tabelle listet alle in diesem Dokument verwendeten QNames auf:

Präfix	Namespace URI
rdf	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a>
rdfs	<a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a>
dc	<a href="http://purl.org/dc/elements/1.1/">http://purl.org/dc/elements/1.1/</a>
i	imaginärer Namespace für Beispiele

Tab. 1: Definition der verwendeten Präfixe

Den Abschluss dieser Einführung in RDF bilden die drei verschiedenen Repräsentationsmodi von RDF Statements, die jeweils anhand eines Beispiels verdeutlicht werden.

#### 3.1. Tripelschreibweise

Diese Schreibweise repräsentiert jedes RDF Statement als ein geordnetes Tripel der Form:

**(Subjekt, Prädikat, Objekt)**

Zusätzlich gelten die folgenden Vereinbarungen für die Kennzeichnungen der einzelnen Elementarten. Doppelte Hochkommas signalisieren, dass der Text zwischen ihnen einem Literal entspricht, während die Schreibweise `<I>` andeutet, dass der Eintrag der Ressource entspricht, die mit dem ausgeschriebenen URI `I` identifiziert wird. Außerdem ist im Folgenden der Präfix `_:` blank Nodes vorangestellt.

Bei dieser Repräsentationsform sollte man sich stets vor Augen halten, dass eine RDF Datenbank einer Menge von Tripel entspricht und es somit keine doppelten Einträge gibt.

### Beispiel 1:

Die Aussage, dass die mit der URI `http://www.example.org` identifizierte Ressource eine Homepage ist und von einem Objekt mit dem Namen "JohnSmith" kreiert wurde, entspricht den folgenden drei RDF Statements:

```
( <http://www.example.org>, rdf:type, <http://www.example.org#homepage> )
( <http://www.example.org>, dc:creator, _:JohnSmith )
( _:JohnSmith, i:name, "JohnSmith" )
```

## 3.2. Graphdarstellung

Die Darstellung mittels *RDF Graphen* ermöglicht es auf intuitive Weise den Zusammenhang zwischen einer überschaubaren Menge von RDF Statements zu erlangen. Ein RDF Tripel entspricht dabei genau einer gerichteten Kante, die mit dem Prädikat beschriftet ist.

Ressourcen werden bei dieser Darstellungsform mittels Ovalen repräsentiert, wobei diese für explizite Objekte mit deren URI beschriftet werden, während blank Nodes leeren Ovalen entsprechen. Literale hingegen werden durch Rechtecke dargestellt.

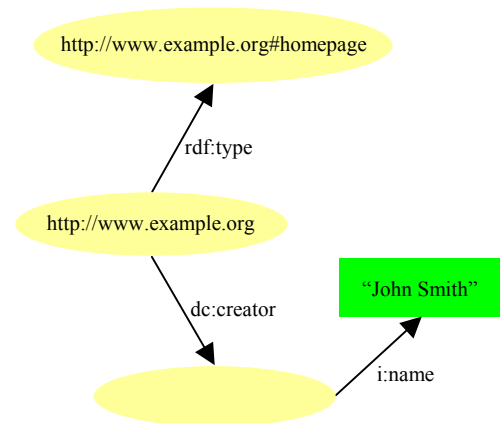


Abb. 1: RDF Graph für Beispiel 1

## 3.3. XML/RDF

Diese Art der Kodierung dient dem Ziel der *Serialisierung* und des Austauschs von RDF Statements, wobei durch die Verwendung von XML eine Schnittstelle geschaffen wurde, die die Kompatibilität unabhängig entwickelter Applikationen maximieren soll. Eine genaue Einführung in XML/RDF kann [1] entnommen werden.

### Beispiel 2:

Die serialisierte Form der im Beispiel 1 beschriebenen Tripel könnte wie folgt aussehen:

```
<?xml version="1.0" ?>
<!DOCTYPE rdf:RDF [<!ENTITY rdfs "http://www.w3.org/2001/SMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:i="http://www.example.org/elements/">
  <rdf:Description rdf:about="http://www.example.org">
    <rdf:type rdf:resource="http://www.example.org#homepage"/>
    <dc:creator rdf:nodeID="JohnSmith"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="JohnSmith">
    <i:name rdf:datatype="&rdfs:string">John Smith</i:name>
  </rdf:Description>
</rdf:RDF>
```

## 4. Grundlegende Definitionen

Nach der vorangegangenen Einführung in RDF folgt in diesem Abschnitt eine formale Charakterisierung von RDF Graphen. Zudem werden die für die Analyse von Anfragen auf RDF Datenbanken entscheidenden Begriffe der Vererbung und Kompaktheit von RDF Graphen definiert.

### 4.1. RDF Graphen

Ein RDF Graph ist ein gerichteter Graph, dessen Kanten und Knoten benannt sind. Es sind Multi-kanten erlaubt, wobei diese der Einschränkung unterliegen, dass alle Kanten zwischen zwei Knoten eindeutige Namen tragen müssen. Des Weiteren kann der Graph zyklisch sein und aus mehreren Zusammenhangskomponenten bestehen.

Für die weiteren Definitionen sowie die Analyse von RDF Graphen werden drei disjunkte Mengen benötigt. Zunächst repräsentiert  $U$  die Menge aller RDF URI Referenzen,  $B$  die Menge aller anonymen Ressourcen und  $L$  die Menge aller RDF Literale. Zusätzlich entspricht  $R$  (Menge aller Ressourcen) der Vereinigungsmenge von  $U$  und  $B$ . Für eine einfachere Schreibweise wird noch mit  $UBL$  die Vereinigung der drei Grundmengen  $U$ ,  $B$  und  $L$  abgekürzt.

Die Knotenmenge  $V$  des Graphen entspricht der Vereinigungsmenge der Ressourcen und der Literale. Für jeden Knoten, der keiner anonymen Ressource entspricht, ist die Funktion  $ID$  definiert als:

$$ID : V \rightarrow U \cup L$$

Für die Unterscheidung von anonymen Ressourcen wird noch eine zusätzliche nur intern verwendete KnotenID definiert, die jedem Knoten einen innerhalb des RDF Graphen eindeutigen Identifier zuordnet.

Die Kanten symbolisieren die Properties der beschriebenen Ressourcen und entsprechen somit den Prädikaten der RDF Statements. Sie sind dabei vom Subjekt zum Objekt gerichtet.

#### Bemerkung 1:

Eine unmittelbare Konsequenz der obigen Festlegung ist, dass anonyme Ressourcen nur mittels Properties abgefragt werden können, die auf sie verweisen.

#### Definition 1:

1. Ein Tripel  $T = (s, p, o)$  nennt man *RDF Tripel*, wenn  $T \in R \times U \times UBL$  gilt.
2. Ein *RDF Graph*  $G$  wird durch die Menge seiner Kanten definiert, die mittels RDF Tripeln repräsentiert werden. Des Weiteren bildet jede Teilmenge dieser Tripel einen Teilgraphen von  $G$ .
3. Ein Graph, der frei von anonymen Ressourcen ist, wird *einfach (ground)* genannt.

#### Definition 2:

1. Zwei Knoten, die keinen anonymen Ressourcen entsprechen, sind *äquivalent* wenn ihre URI gleich ist. Zwei anonyme Ressourcen werden als identisch angesehen, wenn ihre Properties gleich sind und zudem auch äquivalente Ausprägungen aufweisen.
2. Zwei RDF Graphen sind *isomorph*, wenn sie sich lediglich in den KnotenIDs der anonymen Ressourcen unterscheiden.
3. Unter dem *Mergen zweier RDF Graphen*  $G_1$  und  $G_2$  versteht man den RDF Graphen, der der Vereinigung der beiden Tripelmengen entspricht, wobei zuvor mittels einer adäquaten Umbenennung der KnotenIDs von anonymen Ressourcen Namenskonflikte vermieden werden.

Die unter Drittens erwähnte Umbenennung wird dadurch realisiert, dass  $G_1$  mit einem Graphen  $G_2'$  vereinigt wird, der zu  $G_2$  isomorph ist und keine gemeinsamen KnotenIDs für anonyme Ressourcen mit  $G_1$  teilt. Zudem folgt unmittelbar, dass das Ergebnis des Mergens eindeutig ist.

## 4.2. Pseudographen

Für den Vergleich von RDF Graph mit anderen Graphen wird der Begriff des Pseudographen eingeführt, der eine präzisere Charakterisierung ermöglicht.

### Definition 3:

1. Ein *Pseudograph* ist ein Viertupel  $(V, E, l_V, l_E)$  mit den beiden endlichen Mengen  $V$  (Knoten) und  $E$  (Kanten) sowie den beiden Bezeichnungsfunktionen  $l_V, l_E$ .
2. Zwei Pseudographen  $G_1 = (V_1, E_1, l_{V_1}, l_{E_1})$  sind *isomorph* wenn ein Graphisomorphismus  $\Phi: G_1 \rightarrow G_2$  existiert, für den zusätzlich  $\Phi \circ l_{V_1} = l_{V_2} \circ \Phi$  sowie  $\Phi \circ l_{E_1} = l_{E_2} \circ \Phi$  gilt.

Mit der oberen Definition entsprechen Pseudograph gerichteten Graphen die Eigenschleifen (Zyklen) und Mehrfachkanten erlauben. Aus diesem Grund lässt sich ein RDF Graph als ein spezieller Pseudograph definieren, bei dem  $V := UBL$  sowie  $E := U$  gesetzt werden. Zudem gelten die beiden Einschränkungen, dass Knoten, in denen Kanten beginnen keinen Literalen entsprechen dürfen und dass das Bild( $l$ ) in  $U$  enthalten sein muss.

### Korrolar 1:

Zwei RDF Graphen  $G_1, G_2$  sind isomorph wenn ein Isomorphismus für Pseudographen  $\Phi: G_1 \rightarrow G_2$  existiert, der folgende Eigenschaft erfüllt,  $\forall u \in U : \Phi(u) = u$ , d.h. für URI Referenzen entspricht der Isomorphismus der Identitätsabbildung.

### Bemerkung 2:

Aus der vorherigen Charakterisierung von RDF Graphen wird unmittelbar ersichtlich, dass man mit ihnen Standardgraphen wie folgt kodieren kann. Für den Graphen  $G = (V, E)$  benötigt man hierzu:

1.  $u \in U$  (URI, die  $G$  eindeutig zugewiesen wird)
2.  $A \subset B : |A| = |B|$  (anonyme Ressourcen zur Knotenkodierung)
3.  $c : V \rightarrow A : c$  bijektiv (Bijektion zwischen  $V$  und  $A$ )

Dann wird der ursprüngliche Graph durch die folgende Menge von RDF Tripeln kodiert:

$$G_{\text{RDF}} = \{(c(i), u, c(j)) \mid (i, j) \in E\}$$

## 4.3. Interpretation von RDF Graphen

Der folgende Abschnitt orientiert sich an den Definitionen des RDF Semantik Dokuments [2] und dient der Einführung von einfachen *Interpretationen*, d.h. von Interpretationen eines RDF Graphen, die nicht auf den Bedeutungen basieren, die einem speziellen Vokabular angeheftet sind. Eine Interpretation ist hierbei definiert als eine Menge von Annahmen, die die Aussagen des RDF Graphen nicht falsifizieren.

Das Ziel der folgenden Betrachtung ist die Motivation des Begriffs der *Vererbung (Entailment)*, der durch das *Interpolations Lemma* charakterisiert wird. Ihm wird besondere Bedeutung beigemessen, da er die Beziehung des Enthaltenseins von Aussagen, die RDF Graphen innewohnen, beschreibt und somit ein Bindeglied zwischen der modell-theoretischen Semantik von RDF und realen Applikationen bildet.

Zusammenfassend heißt dies, dass die Aussage  $G$  vererbt an  $G'$  äquivalent damit ist, dass jede Interpretation, die  $G$  wahr macht ebenso auch  $G'$  wahr macht und somit jede Behauptung über  $G$  eine gewisse Aussage über  $G'$  impliziert.

Für den Anfang wird zunächst die Definition einer einfachen Interpretation betrachtet:

**Definition 4:**

Eine *einfache Interpretation*  $I$  eines Vokabulars  $V$  besteht aus:

1. Einer nicht leeren Menge  $\mathbb{R}$  von Ressourcen, die der Domäne oder dem Universum von  $I$  entspricht.
2. Einer Menge  $\mathbb{P}$ , die alle Properties von  $I$  enthält.
3. Einer Abbildung  $\text{IEXT} : \mathbb{P} \rightarrow \{x \mid x \subset \mathbb{R} \times \mathbb{R}\}$   
 $\text{IEXT}(p)$  wird als extension von  $p$  bezeichnet und umfasst die Menge aller Paare, für die die binäre Eigenschaft  $x$  wahr ist.
4. Einer Abbildung  $\text{IS} : U \rightarrow V$  ( $V \equiv \mathbb{R} \cup \mathbb{P}$ )
5. Einer Abbildung  $\text{IL}$  von den typisierten Literalen in  $V$  nach  $\mathbb{R}$ .
6. Einer ausgewählten Teilmenge  $\mathbb{L}$  von  $\mathbb{R}$ , die zumindest alle Strings sowie allen Paaren bestehend aus Strings und Sprach-Tags enthält.

Bevor nun die rekursiv definierte Bedeutung eines RDF Graphen eingeführt werden kann, bedarf es des Begriffs des Mappings.

**Definition 5: Mapping**

1. Eine Abbildung  $\mu : \text{UBL} \rightarrow \text{UBL}$ , die für alle Literale und URI Referenzen der Identitätsfunktion entspricht, d.h. diese Parametertypen bewahrt, wird ein *Mapping* genannt.
2. Für einen gegebenen Graphen  $G$  wird  $\mu(G)$  definiert als:
 
$$\mu(G) = \{(\mu(s), \mu(p), \mu(o)) \mid (s, p, o) \in G\}$$
3. Ein Mapping  $\mu$  ist *konsistent* wenn  $\mu(G)$  wieder einem RDF Graphen entspricht, d.h. für alle Elemente  $(\mu(s), \mu(p), \mu(o)) \in \mathbb{R} \times U \times \text{UBL}$  gilt. Ist dies der Fall, dann heißt  $\mu(G)$  eine *Instanz* des ursprünglichen Graphen  $G$ .
4. Von einer *echten Instanz* ist die Rede, wenn die Anzahl der anonymen Ressourcen in  $\mu(G)$  geringer ist als die in  $G$ . Dies kann durch die Instanziierung oder das Zusammenlegen zwei anonymer Ressourcen geschehen.

**Definition 6: Graphinterpretation**

Die Interpretation eines Graphen ist wie folgt rekursiv definiert:

- $X$  ist ein einfaches Literal "Text"  $\rightarrow I(X) \equiv \text{Text}$
- $X$  ist ein einfaches Literal "Text"@ $xy$   $\rightarrow I(X) \equiv \langle \text{Text}, xy \rangle$
- $X$  ist ein typisiertes Literal aus  $V$   $\rightarrow I(X) \equiv \text{IL}(X)$
- $X$  ist eine URI Referenz aus  $V$   $\rightarrow I(X) \equiv \text{IS}(X)$
- $X$  ist ein einfaches Tripel  $\langle s, p, o \rangle$   $\rightarrow$  aus  $s, p, o \in V$ ,  $I(p) \in \mathbb{P}$  und  $(I(s), I(o)) \in \text{IEXT}(I(p))$  folgt  $I(X) \equiv \text{true}$ , ansonsten gilt  $I(X) \equiv \text{false}$
- $X$  ist ein einfacher RDF Graph  $\rightarrow$  wenn  $I(X) \equiv \text{false}$  für ein Tripel  $T \in X$ , ansonsten gilt  $I(X) \equiv \text{true}$

Bei der Verwendung von anonymen Ressourcen benötigt man noch zusätzlich:

- $X$  ist eine anonyme Ressource  $\rightarrow [I+\mu](X) \equiv I(\mu(X))$
- $X$  ist ein RDF Graph  $\rightarrow$  wenn  $[I+\mu](X) \equiv \text{true}$  für ein Mapping  $\mu$  mit dem Bild( $\mu$ )  $\mathbb{R}$ , dann folgt  $I(X) \equiv \text{true}$  ansonsten gilt  $I(X) \equiv \text{false}$

### Definition 7: Subinterpretation

Eine Interpretation  $I$  ist eine *Subinterpretation*, dargestellt durch  $I \ll J$ , einer zweiten Interpretation  $J$  wenn eine Abbildung  $k : (\mathbb{R}_I \cup \mathbb{IP}_I) \rightarrow (\mathbb{R}_J \cup \mathbb{IP}_J)$  mit folgenden Eigenschaften existiert:

1.  $x \in \mathbb{R}_I \rightarrow k(x) \in \mathbb{R}_J$
2.  $x \in \mathbb{IP}_I \rightarrow k(x) \in \mathbb{IP}_J$
3.  $x$  ist ein String oder ein Paar von Strings  $\rightarrow k(x) = x$
4.  $x \in (U \cup L) \rightarrow J(x) = k(I(x))$
5.  $(x, y) \in \text{IEXT}_I(p) \rightarrow (k(x), k(y)) \in \text{IEXT}_J(k(p))$

### Lemma 1: Subinterpretations Lemma

Aus  $I \ll J$  und  $I(G) \equiv \text{true}$  folgt  $J(G) \equiv \text{true}$ .

#### Beweis:

Nach Voraussetzung existiert ein Mapping  $\mu$ , so dass  $[I + \mu]((s, p, o)) \equiv \text{true}$  für alle  $(s, p, o) \in G$  gilt.

$\rightarrow ([I + \mu](s), [I + \mu](o)) \in \text{IEXT}_I(I(p))$

$\rightarrow$  für die Abbildung  $k$  der Subinterpretation gilt dann  $(k([I + \mu](s)), k([I + \mu](o))) \in \text{IEXT}_J(k(I(p)))$

$\rightarrow$  Sei  $\Phi(x) := k(\mu(x))$  und  $[J + \Phi](x) \equiv k([I + \mu](x))$  dann gilt  $([J + \Phi](s), [J + \Phi](o)) \in \text{IEXT}_J(J(p))$

$\rightarrow J(G) \equiv \text{true}$

### Definition 8: Herbrand Interpretation

Die *Herbrand Interpretation* eines Graphen  $G$ , abgekürzt durch  $\text{Herb}_G$  ist wie folgt definiert:

- $\text{LV}$  ist die Menge aller in  $G$  vorkommenden Literale
- $\mathbb{R}$  beinhaltet  $\text{LV}$  sowie alle Namen und alle anonymen Ressourcen, die als Objekte oder Subjekte in Tripeln aus  $G$  auftreten
- $\mathbb{IP}$  beinhaltet alle URI Referenzen, die als Properties in  $G$  verwendet werden
- $\text{IS}$  ist die Identitätsabbildung für das Vokabular von  $G$
- $\text{IL}$  ist die Identitätsabbildung für alle typisierten Literale
- $\text{IEXT}$  wird definiert als  $(s, o) \in \text{IEXT}(p) \leftrightarrow (s, p, o) \in G$

Aus dieser Definition folgt unmittelbar, dass  $\text{Herb}_G + \mu$ , mit der Identitätsabbildung auf den anonymen Ressourcen als  $\mu$ , alle Tripel in  $G$  erfüllt und deshalb  $\text{Herb}_G(G) \equiv \text{true}$  gilt.

### Definition 9: einfache Vererbung

Ein RDF Graph  $G_1$  *vererbt* an einen weiteren RDF Graphen  $G_2$ , dargestellt durch  $G_1 \models G_2$ , wenn jede Interpretation  $I$  des Vokabulars von  $(G_1 \cup G_2)$  mit  $I(G_1) \equiv \text{true}$  auch  $I(G_2) \equiv \text{true}$  impliziert.

Nach diesen grundlegenden Definitionen kann jetzt das für die Vererbung bedeutende Interpolations Lemma eingeführt werden, dass Vererbung auf einfache Beziehungen unter Graphen zurückführt. Zuvor werden jedoch noch drei Lemmata von Herbrand betrachtet, deren unmittelbare Konsequenz das Interpolation Lemma ist.

### Lemma 2: Herbrand Lemma

$I(G) \equiv \text{true} \leftrightarrow \text{Herb}_G \ll I$

**Beweis:**

- :  $I(G) \equiv \text{true}$
- man wähle einen Teil von  $\text{Herb}_G$  als Abbildung  $V_{\text{Herb}_G} \rightarrow V_I$
  - $I(\text{Herb}_G(a)) = I(a) \wedge \forall (s, p, o) \in G \text{ gilt } (s, o) \in \text{IEXT}_{\text{Herb}_G}(\text{Herb}_G(p))$
  - da  $(I(s), I(o)) \equiv (I(\text{Herb}_G(s)), I(\text{Herb}_G(o)))$  gilt  $(I(\text{Herb}_G(s)), I(\text{Herb}_G(o))) \in \text{IEXT}_I(I(p))$
  - $(I(\text{Herb}_G(s)), I(\text{Herb}_G(o))) \in \text{IEXT}_{I(\text{Herb}_G)}(I(\text{Herb}_G(p)))$ , was der Forderung an die subinterpretations Abbildung  $\text{Herb}_G \rightarrow I$  entspricht.
  - $\text{Herb}_G \ll I$
- ←:  $\text{Herb}_G \ll I$
- per Definition gilt  $\text{Herb}_G \equiv \text{true}$
  - mit dem Subinterpretations Lemma gilt  $I(G) \equiv \text{true}$

**Lemma 3: Herbrand Entailment Lemma**

$$G \models G' \leftrightarrow \text{Herb}_G(G') \equiv \text{true}$$

**Beweis:**

- :  $G \models G' \wedge \text{Herb}_G(G) \equiv \text{true}$
- $\exists \mu : \mu(G') \subseteq G$
  - $[\text{Herb}_G + \mu](G') \equiv \text{true}$
  - $\text{Herb}_G(G') \equiv \text{true}$
- ←:  $\text{Herb}_G(G') \equiv \text{true}$
- $\forall I : I(G) \equiv \text{true}$  gilt  $\text{Herb}_G \ll I$
  - $I(G') \equiv \text{true}$
  - $G \models G'$

**Definition 10:**

Seien  $G$  und  $G'$  RDF Graphen, dann ist  $G'$  mit  $G$  verbunden, wenn eine Instanz von  $G'$  ein Teilgraph von  $G$  ist. Insbesondere ist ein einfaches Tripel  $T$  mit  $G$  verbunden wenn  $T \in G$  gilt.

**Lemma 4: Herbrand Separations Lemma**

$$\text{Herb}_G(G') \equiv \text{true} \leftrightarrow G' \text{ ist mit } G \text{ verbunden}$$

**Beweis:**

- $\text{Herb}_G(G') \equiv \text{true}$
- ↔  $\exists \mu : [\text{Herb}_G + \mu](G') \equiv \text{true}$
  - ↔  $\exists \mu : \mu(G') \subseteq G$
  - ↔  $G'$  ist mit  $G$  verbunden

Die Kombination der Implikationen aus dem Entailment- und dem Separation Lemmas von Herbrand führt unmittelbar auf das Interpolation Lemma, das für die weiteren Betrachtungen von großer Bedeutung ist:

**Lemma 5: Interpolation Lemma**

Seien  $G_1$  und  $G_2$  zwei RDF Graphen. Dann gilt  $G_1 \models G_2$ , genau dann wenn eine Instanz von  $G_2$  ein Teilgraph von  $G_1$  ist.

**Beispiel 3:**

1. Ein Graph  $G$  vererbt trivialerweise an alle seine Teilgraphen.
2. Für die folgenden drei Graphen gilt  $G_1 \models G_3$  aber  $G_1 \not\models G_2$

$$G_1 = \{ (a, b, c), (\_ :X, b, c), (a, b, \_ :Y) \}$$

$$G_2 = \{ (\_ :U, \_ :V, c), (\_ :V, b, c) \}$$

$$G_3 = \{ (a, \_ :V, c), (\_ :X, b, \_ :Y) \}$$

**Korollar 2:**

Zwei Graphen  $G_1$  und  $G_2$  sind *äquivalent*, dargestellt durch  $G_1 \equiv G_2$ , wenn  $G_1 \models G_2$  und  $G_2 \models G_1$  gilt.

**4.4. Minimale Repräsentation**

Durch die Verwendung von anonymen Ressourcen können *Redundanzen* innerhalb eines RDF Graphen auftreten, deren Vermeidung wünschenswert ist. Die Frage nach einer *minimalen Repräsentation* von Daten ist unmittelbar an den Begriff des kompakten Graphen gebunden.

Dieser Abschnitt beschäftigt sich mit Graphen, die kompakt sind, wobei schon die Definition dieser Graphenart unterstreicht, dass es sich um Graphen handelt, die frei von Redundanzen sind.

**Definition 11: Kompaktheit von Graphen**

Ein Graph  $G$  wird *kompakt* genannt, wenn kein Mapping  $\mu$  existiert, so dass  $\mu(G)$  ein echter Teilgraph von  $G$  ist, d.h. jede echte Instanz von  $G$  ist kein Teilgraph von  $G$ .

**Beispiel 4:**

Seien die beiden RDF Graphen  $G_1$  und  $G_2$  wie folgt definiert

$$G_1 = \{ (a, b, c), (\_ :X, b, c) \}$$

$$G_2 = \{ (a, b, c), (\_ :X, b, c), (\_ :X, c, d) \}$$

dann ist  $G_2$  kompakt während  $G_1$  diese Eigenschaft nicht besitzt, d.h. die Eigenschaft der Kompaktheit eines Graphen ist unter Teilmengenbildung nicht abgeschlossen.

Das folgende Lemma zeigt, dass mit der Hilfe des im vorherigen Abschnitt eingeführten Interpolations Lemmas unmittelbare Konsequenzen für die Vererbung zwischen zwei RDF Graphen getroffen werden können, wenn letztere in bestimmten Verhältnissen zueinander stehen.

**Lemma 6: Anonymity Lemma**

1. Ein kompakter Graph vererbt nicht an seine echten Instanzen.
2. Wenn  $G'$  aus einem kompakten Graphen  $G$  durch Zusammenlegung zweier anonymer Ressourcen hervorgeht, dann vererbt  $G$  nicht an  $G'$ .

**Beweis:**

1. Sei  $G$  ein kompakter Graph und  $G'$  eine echte Instanz von  $G$ . Des Weiteren gilt  $G \models G'$ , d.h.  $G$  vererbt an  $G'$ 
  - $\exists$  ein konsistentes Mapping  $\mu$ , so dass die Instanz  $\mu(G')$  ein Teilgraph von  $G$  ist
  - $\exists$  ein weiteres konsistentes Mapping  $\psi : \psi(G)$  ist Teilgraph von  $G$ , da die Eigenschaft Instanz von transitiv ist
  - $G$  kann nicht kompakt sein
  - ein kompakter Graph kann nicht an seine echten Instanzen vererben
2. Sei  $G$  ein kompakter Graph und gehe  $G'$  durch Zusammenlegung zweier anonymer Ressourcen aus  $G$  hervor
  - $G'$  kann kein Teilgraph von  $G$  sein
  - keine Instanz von  $G'$  ist ein Teilgraph von  $G$ , da die Eigenschaft Instanz von transitiv ist
  - $G \not\models G'$

Ansonsten sind die Eigenschaften von kompakten Graphen zu großen Teilen noch unerforscht und bedürfen deshalb einer eingehenden Betrachtung. Zwei grundlegende Aspekte, die im Folgenden beantwortet werden, sind die Frage nach einer eindeutigen Zuordnung von kompakten Graphen sowie der Komplexität, die der Bestimmung eines solchen Graphen innewohnt.

Die beiden folgenden Sätze widmen sich diesen Fragestellungen.

**Satz 1:**

Für jeden RDF Graph  $G$  existiert ein *eindeutig bestimmter kompakter Graph*, zu dem  $G$  äquivalent ist.

**Beweis:**

Zunächst definieren wir innerhalb der Menge der RDF Graphen die zweistellige Relation  $\rightarrow$ . Es gilt  $G_1 \rightarrow G_2$ , wenn  $G_1$  ein echter Teilgraph von  $G_2$  ist und ein konsistentes Mapping  $\mu$  existiert, so dass  $G_2 \equiv \mu(G_1)$  gilt.

Für den Beweis werden von der Relation zwei Eigenschaften gefordert. Erstens darf sie für den Nachweis der Existenz von kompakten Graphen keine endlosen Ketten von Hintereinanderausführungen besitzen. Und Zweitens müssen die Endgraphen, die beim Auftreten von Wahlmöglichkeiten resultieren, äquivalent zueinander sein, so dass die Eindeutigkeit des kompakten Graphen gewährleistet ist.

Die erste Forderung folgt unmittelbar aus der Definition der Relation, da nur endliche Graphen verwendet werden. Für die Zweite muss man den Fall  $G_1 \leftarrow G \rightarrow G_2$  betrachten, für den der transitive Abschluss  $\rightarrow^*$  mit der Eigenschaft  $G_1 \rightarrow^* G^*$  und  $G_2 \rightarrow^* G^*$  benötigt wird. Diese Eigenschaft wird jedoch von der Relation gewährleistet, wie die folgende Überlegung zeigt.

Angenommen es gebe kein eindeutiges  $G^*$  sondern zwei kompakte Graphen  $G_1^*$  und  $G_2^*$

- O.B.d.A.  $\exists (s, p, o) \in G_1^*$  aber  $(s, p, o) \notin G_2^*$
- $\exists (s', p, o') \in G_2^*$  in das  $(s, p, o)$  in ein oder mehreren Schritten gemappt wurde, da in beiden Hintereinanderausführungen vom gleichen Graphen  $G$  ausgegangen wurde
- $\exists \mu(G_1)$  so dass  $\mu(G_1)$  ein echter Teilgraph von  $G_1^*$  ist
- $G_1^*$  kann nicht kompakt sein!

Aus dem obigen Argument folgt, dass die Relation  $\rightarrow$  konvergiert und deshalb für jeden Graphen  $G$  ein eindeutiges  $G^*$  existiert, so dass  $G \rightarrow^* G^*$  gilt und  $G^*$  im Sinne der Relation irreduzibel ist. D.h.  $G^*$  ist der gewünschte kompakte Graph.

Nachdem die eindeutige Zuordnung von kompakten Graphen sichergestellt ist, stellt sich nun die Frage nach einer effizienten Berechnung, die in dem folgenden Satz beantwortet wird.

**Satz 2:**

Die Beantwortung der Frage, ob ein RDF Graph kompakt ist, ist NP-vollständig.

**Beweis:**

Für die Motivation des Beweises wird an die Bemerkung 2 (aus 4.2) erinnert, in der gezeigt wird, dass jeder Graph in Form eines RDF Graphen, dessen Knoten nur aus anonymen Ressourcen besteht, dargestellt werden kann. Da die dort gegebene Konstruktionsvorschrift bijektiv ist, lässt sie sich eindeutig umkehren und man erhält bereits für diese spezielle Teilmenge von RDF Graphen die NP-Vollständigkeit der Frage, ob ein solcher RDF Graph kompakt ist.

Der Grund hierfür ist, dass die obige Frage äquivalent zu dem Problem  $\text{CORE}_G$  ist, in dem für einen gegebenen Graphen  $G$  nach der Existenz eines nicht surjektiven Endomorphismus  $G \rightarrow G$  gefragt wird. Zusätzlich wurde von Hell und Nestril in [6] nachgewiesen, dass das Problem  $\text{CORE}_G$  NP-vollständig ist.

Eine unmittelbare Aussage des letzten Satzes ist, dass die minimale Repräsentation eines RDF Graphen schwer zu berechnen ist.

## 5. Anfragen auf RDF Datenbanken

Durch die Definition eines RDF Graphen als eine Menge von RDF Tripeln kann dieser als eine *relationale Datenbank* interpretiert werden, die eine Relation mit den drei Attributen Subjekt, Prädikat und Objekt enthält. Dabei entsprechen die Tripel des RDF Graphen den Tupeln dieser Relation.

Der einzige Unterschied zu einer relationalen Datenbank, ergibt sich durch die Anwesenheit von anonymen Einträgen, die im Gegensatz zu nur einem NULL Wert eindeutig unterschieden werden können und zudem auch noch Properties besitzen.

Somit wird im Folgenden eine RDF Datenbank als ein RDF Graph angesehen.

### 5.1. Abfragesprache

Für die Formulierung von Abfragen wird eine Menge von *Variablen*  $V$ , die disjunkt mit UBL ist, eingeführt. Im Folgenden werden Variablen durch Bezeichner dargestellt, die den Präfix „?“ aufweisen. Die hier verwendete Abfragesprache bedient sich des in der Datenbankliteratur weit verbreiteten Begriffs des *Tableaus* [7], der wie folgt modifiziert wird:

**Definition 12: Abfrage**

Eine *Abfrage* wird definiert als ein Tripel  $(H, B, C)$ , das aus einem *Header*  $H$ , einem *Body*  $B$  und einer Menge von *Einschränkungen*  $C$  (Constraints) besteht. Für die Elemente bestehen folgende Beziehungen:

1.  $H$  ist ein RDF Graph über  $V \cup \text{UBL}$ , mit der Eigenschaft  $\text{var}(H) \subseteq \text{var}(B)$
2.  $B$  ist ein RDF Graph über  $V \cup \text{UL}$
3.  $C \subseteq \text{var}(H)$

Der Body dient dabei der Einschränkung der Ergebnismenge, indem er nur Variablen zulässt, die bestimmte Properties aufweisen. Diese werden dann auf die im Header definierte Formatierung gemappt, weshalb oft die Schreibweise  $H \leftarrow B$  Verwendung findet. Durch die Menge  $C$  wird die Ergebnismenge dahingehend eingeschränkt, dass alle Variablen, die in  $C$  auftreten nicht mit anonymen Ressourcen belegt sein dürfen.

**Beispiel 5:**

Die folgende Abfrage, die keine Einschränkungen besitzt, würde einfach alle Personen die eine Rechnung bezahlen müssen, als solche definieren, die ein Telefon besitzen, dem diese Rechnung zugeschrieben wird.

$$(?Person, \text{pays}, ?Bill) \leftarrow (?Person, \text{owns}, ?Phone), (?Phone, \text{causes}, ?Bill)$$

Sinnvoll wäre es noch mittels  $C := \{ ?Person \}$ , die Ergebnismenge auf alle Personen einzuschränken, die mittels einer nicht anonymen Ressource identifiziert werden können.

**Bemerkung 4:**

Durch die Bedingung  $\text{var}(H) \subseteq \text{var}(B)$  wird verhindert, dass ungebundene Variablen im Header der Abfrage auftauchen können. Des Weiteren sind anonyme Ressourcen im Body einer Anfrage redundant, da sich zeigen wird, dass sie in diesem Abfrageteil äquivalent zu Variablen sind. Hingegen ist ihre Anwesenheit im Header wünschenswert, da somit Reification (auf die in 5.3 näher eingegangen wird) mittels Abfragen ermöglicht wird. Technisch gesehen entsprechen sie dort freien Termen der Form  $f(X_1, \dots, X_n)$ , wobei die  $X_1, \dots, X_n$  einer Teilmenge der in der Abfrage auftretenden Variablen bzw. Konstanten entsprechen.

**5.2. Antworten auf Anfragen**

Die *Antwort* auf eine Anfrage entspricht einer Belegung der Variablen, so dass alle Bedingungen erfüllt sind. Für eine formale Definition wird der Begriff der *Bewertung* benötigt. Darunter versteht man eine Abbildung  $v : V \rightarrow \text{UBL}$ . Falls die Anfrage Einschränkungen besitzt, d.h. die Menge  $C$  nicht leer ist, dann werden diese von der Bewertung  $v$  erfüllt (dargestellt durch  $v \models C$ ), wenn für alle  $x \in C$ ,  $v(x)$  keiner anonymen Ressource entspricht.

Unter einem Matching eines Graphen  $G$  in einer Datenbank  $D$  versteht man eine Bewertung  $v$ , so dass eine Instanz von  $v(B)$  ein Teilgraph von  $D$  ist, insbesondere heißt das, dass  $D \models v(B)$  gilt. Von besonderem Interesse sind natürlich die Matchings, die die Einschränkungen erfüllen.

**Definition 13:**

Für eine *Anfrage*  $q = (H, B, C)$  auf eine Datenbank  $D$  definiert man ihre *Vorantwort* (pre-answer) wie folgt als eine Menge von RDF Graphen:

$$\text{preans}(q, D) = \{ v(H) : D \models v(B) \wedge v \models C \}$$

Dabei bezeichnet man jeden Graph  $v(H)$  als eine einfache Antwort (single answer) der Anfrage  $q$  auf die Datenbank  $D$ .

Für die Kombination aller einfachen Antworten zu der endgültigen Antwort auf eine Abfrage existieren die folgenden zwei Möglichkeiten, die zu unterschiedlichen Semantiken führen:

1. Die Antwort, die mit  $\text{ans}_u(q, D)$  bezeichnet wird, entspricht der mengentheoretischen Vereinigung (*Union-Semantik*) aller einfachen Antworten. Dieser Ansatz zeichnet sich dadurch aus, dass anonyme Ressourcen zu Bindegliedern zwischen den einzelnen einfachen Antworten werden können und somit die Möglichkeit für die Modellierung von neuen Zusammenhängen in der Antwort besteht.
2. Der alternative Ansatz, dessen Ergebnis mit  $\text{ans}_m(q, D)$  bezeichnet wird entspricht dem Mergen der einzelnen einfachen Antworten (*Merge-Semantik*). Dies hat zur Folge, dass der RDF Graph, der die endgültige Antwort symbolisiert, einem Wald von einfachen Antworten entspricht.

Für den Fall, dass die Datenbank  $D$  ein einfacher RDF Graph ist, d.h. keine anonymen Ressourcen besitzt sind beide Ansätze äquivalent, da diese Situation einer Anfrage auf eine klassische Datenbank entspricht.

**Beispiel 6:**

Die beiden hier aufgelisteten Fälle skizzieren Szenarien, in denen die Union-, der Merge-Semantik überlegen ist, weshalb in den folgenden Abschnitten stets von der Union-Semantik ausgegangen wird, falls nichts anderes vermerkt wird.

1. Mit der Merge-Semantik ist eine datenunabhängige Identitätsabfrage nicht möglich, während mit der Union-Semantik für die Abfrage  $q = (\{ (?s, ?p, ?o) \}, \{ (?s, ?p, ?o) \}, \emptyset)$  und einer beliebigen Datenbank  $D$  stets  $\text{ans}_u(q, D) \equiv D$  gilt.
2. Wenn in einer Datenbank eine anonyme Ressource existiert, die das Objekt eines Statements  $(s, p \_ :B)$  ist und diese anonyme Ressource selber mehrere Properties besitzt, d.h. es existieren mehrere Einträge der Form  $(\_ :B, p_i, o_i) : i \in [1..n]$ , dann ist es nicht möglich alle Properties von  $\_ :B$  mit einer datenunabhängigen Abfrage zu ermitteln, wenn man von der Merge-Semantik ausgeht. Die folgende Abfrage  $q = (\{ (?s, \text{„feature“}, ?p) \}, \{ (s, p, ?s), (?s, ?p, ?o) \}, \emptyset)$  führt hingegen bei der Verwendung der Union-Semantik zum Erfolg.

**Behauptung 1:**

Für beliebige Datenbanken  $D$  und  $D'$  sowie eine beliebige Anfrage  $q$  gilt für beide Semantiken:

$$D \models D' \rightarrow \text{ans}(q, D) \models \text{ans}(q, D')$$

**Beweis:**

Nach Voraussetzung existiert ein Mapping  $\mu$ , so dass  $\mu(D')$  ein Teilgraph von  $D$  ist. Des Weiteren sei  $H$  der Header und  $B$  der Body der Anfrage.

- $\rightarrow \forall v(H) \in \text{preans}(q, D') \exists v(B) : v(B) \subseteq D'$
- $\rightarrow$  für dieses  $v(B)$  gilt wiederum  $\mu(v(B)) \subseteq D$
- $\rightarrow \mu(v(H)) \in \text{preans}(q, D)$
- $\rightarrow \forall G \in \text{preans}(q, D') \exists G' \in \text{preans}(q, D) : G \subseteq G'$
- $\rightarrow \text{ans}(q, D) \models \text{ans}(q, D')$

**Behauptung 2:**

Für alle Anfragen  $q$  und alle Datenbanken  $D$  gilt:

$$\text{ans}_u(q, D) \models \text{ans}_m(q, D)$$

**Beweis:**

Die Aussage folgt unmittelbar aus der Tatsache, dass die Vereinigung zweier Graphen  $G_1, G_2$  an das Ergebnis des Mergens der beiden vererbt. Für eine Überprüfung führe man einfach das Mapping auf dem Ergebnis des Mergens aus, das die Umbenennung der anonymen Ressourcen während des Mergens invertiert.

**Bemerkung 5:**

Die Umkehrung der letzten Behauptung gilt im Allgemeinen nicht. Für ein Gegenbeispiel genügt bereits eine Anfrage, die zwei einfache Antworten besitzt, die eine anonyme Ressource gemeinsam haben, wie z. B. die Identitätsabfrage  $q$  auf die Datenbank  $D = \{ (\_ :X, b, c), (\_ :X, b, d) \}$ . In diesem Fall ist  $\text{ans}_u(q, D) \equiv D$ , während  $\text{ans}_m(q, D) \equiv \{ (\_ :X_1, b, c), (\_ :X_2, b, d) \}$ , so dass natürlich kein Mapping von  $D$  auf  $\text{ans}_m(q, D)$  existiert.

### Bemerkung 6: Redundanz

Für eine effiziente Verarbeitung von Anfragen ist die Vermeidung von Redundanz von großer Bedeutung. Deshalb werden im Folgenden einige Beobachtungen für die Vermeidung von Redundanzen aufgelistet.

1. Abfragen sollten *kompakte Header* besitzen, da ansonsten in der generierten Ergebnismenge Redundanzen auftreten, die vermieden werden könnten.
2. Die Verwendung eines *nicht kompakten Bodies* lässt sich nicht immer vermeiden. Dies ist z.B. immer dann der Fall wenn alle Ausprägungen einer bestimmten Property einer Ressource verlangt werden, wobei nur solche Ressourcen aufgelistet werden sollen, die eine bestimmte Ausprägung dieser Property aufweisen.
3. Selbst wenn die Datenbank sowie die Abfrage kompakt gehalten werden, können in der Ergebnismenge Redundanzen, in Form von nicht kompakten RDF Graphen, auftreten, da die Eigenschaft der Kompaktheit unter Teilmengenbildung nicht abgeschlossen ist (siehe hierzu Bsp. 2)

### 5.3. Reification

Nach den vorherigen Betrachtungen von RDF Graphen, deren Interpretationen sich lediglich auf ihre Syntax beziehen, wird im Folgenden ein Vokabular, d.h. eine Menge von Ressourcen eingeführt, denen eine wohldefinierte Bedeutung angeheftet ist. Dieses Vokabular beschränkt sich auf die folgende, selbsterklärende Teilmenge des *RDF Schemas (RDFS)*, das eine Erweiterung von RDF ist:

rdf:statement	rdf:subject	rdf:predicate
rdf:object	rdf:type	

Durch die Verwendung von RDFS stehen Mechanismen bereit, die es ermöglichen verwandte Ressourcen sowie die Relationen zwischen ihnen zu beschreiben. Jedoch beschränken wir uns mit der obigen Teilmenge auf die Betrachtung von *Reification*. Hierunter versteht man Aussagen über RDF Aussagen, mit denen ein Model des RDF Graphen beschrieben wird, wie z.B. „*Das Tripel (s, p, o) ist ein Statement*“. Diese Aussage lässt sich mit dem oben erwähnten Vokabular durch die folgenden vier Tripel beschreiben, die jedes Element der oben getroffenen Aussage näher charakterisieren:

$$\{ (\_ : \text{Aussage}, \text{rdf:type}, \text{rdf:statement}), (\_ : \text{Aussage}, \text{rdf:subjekt}, s), \\ (\_ : \text{Aussage}, \text{rdf:object}, o), (\_ : \text{Aussage}, \text{rdf:predicate}, p) \}$$

#### Beispiel 7:

Mit der folgenden Abfrage kann die oben beschriebene Reification der Aussage (s, p, o) generiert werden, wobei für die Erzeugung einer eindeutige anonyme Ressource noch eine Skolem Funktion benötigt wird:

$$(\text{f}(a, b, c), \text{rdf:type}, \text{rdf:statement}), (\text{f}(a, b, c), \text{rdf:subjekt}, s), \\ (\text{f}(a, b, c), \text{rdf:object}, s), (\text{f}(a, b, c), \text{rdf:predicate}, p) \quad \leftarrow \quad (s, p, o)$$

Wenn man die Reification, von allen Tripeln in einer Datenbank erzeugen möchte, dann müssen in der oberen Abfrage lediglich alle Vorkommen der Konstanten s, p, o durch die entsprechenden Variablen ?s, ?p, ?o ersetzt werden.

#### Beispiel 8:

Die Ermittlung aller Properties, die einer ausgezeichneten Ressource r innerhalb einer Datenbank angeheftet sind, lässt sich mit folgender Abfrage realisieren:

$$(?Property, \text{rdf:type}, \text{rdf:predicate}) \quad \leftarrow \quad (r, ?Property, ?Object)$$

Damit lässt sich die Leibnitz'sche Identität mittels zweier Abfragen auf eine Datenbank testen, da diese besagt  $x \equiv y \leftrightarrow \forall \text{ Properties } P \text{ gilt } P(x) \equiv P(y)$ .

### Bemerkung 7:

Für die Behandlung von Aussagen bieten sich zwei Möglichkeiten an. Zunächst können Aussagen selbst als Objekte angesehen werden. Dies führt jedoch zu der problematischen Situation, dass bei der Einführung von Reification, jedes Statement selbst wieder ein neues Objekt darstellt, das beschrieben werden kann und somit eine unendliche Menge von Aussagen der Form

$$(\_ : \text{Aussage}_{i+1}, \text{rdf:subject}, \_ : \text{Aussage}_i)$$

gültige Bestandteile einer Datenbank sind. Dies widerspricht jedoch der Definition von RDF Graphen, die als eine endliche Menge definiert sind.

Die zweite Alternative, die dem Vorgehen im RDFS entspricht, betrachtet Aussagen nicht als eigene Objekte, sondern verwendet Namen als Referenzen, was dazu führt, dass die Existenz einer Aussage nicht die Existenz ihrer Reification impliziert sowie vice versa. Ein weiterer Vorteil dieses Vorgehens ist, dass durch die Einführung von Reification, die Größe des RDF Graphen lediglich um einen konstanten Faktor ansteigt.

## 6. Komplexitätsbetrachtungen

Dieser Abschnitt analysiert die *Komplexität*, die der Beantwortung von Anfragen auf eine RDF Datenbank zugrunde liegt. Er ist hierfür in drei Abschnitte unterteilt. Zunächst wird die bloße Beantwortung von Anfragen betrachtet. Die zwei weiteren Abschnitte beschäftigen sich mit *effizienten Antworten*, die kompakten Graphen entsprechen. Dafür werden sowohl die *Minimierung von Anfragen* sowie die *Minimierung von Antworten*, d.h. die Eliminierung von Redundanzen aus der Antwort, betrachtet.

### 6.1. Berechnung von Antworten

Für ein Verständnis der Komplexität, die der Berechnung von Antworten zugrunde liegt, wird ein einfacheres Problem betrachtet, nämlich die Frage, ob die Antwort einer Abfrage leer ist. Zudem erfolgt die Bewertung einer Abfrage  $q$  auf eine Datenbank  $D$  von den folgenden zwei Standpunkten:

1. **Abfrage Komplexität:** Die Datenbank  $D$  ist fixiert, während die Abfrage  $q$  beliebig ist.
2. **Daten Komplexität:** Die Abfrage  $q$  ist fixiert, während die Datenbank  $D$  variabel bleibt.

#### Satz 3:

Das Bewertungsproblem ist NP-hart für die Version Abfrage Komplexität während es in der Version Daten Komplexität in polynomieller Zeit in der Größe der Datenbank lösbar ist.

#### Beweis:

##### 1. Daten Komplexität:

Aus der Reduktion 3-SAT  $\leq_p$  Bewertungsproblem einer konjugierten Abfrage folgt unmittelbar die NP-Härte, da das Erfüllungsproblem bereits selbst NP-vollständig ist. Bei der Reduktion kann dabei wie folgt vorgegangen werden:

In der Datenbank existieren zwei Arten von Ressourcen, Verweisressourcen und Belegungsressourcen.

Bei letzteren entspricht jede Ressource einer eindeutigen Kombination aus einer Klausel der maximalen Länge drei sowie einer Belegung, die sie erfüllt. Zwischen dieser Ressourcenart existiert zusätzlich eine Hand von Properties ( $\text{satisfy}_{\text{Typ}}$ ) die anzeigen, wann zwei solche Klauseln simultan erfüllbar sind. Dabei unterscheiden sich die URI Referenzen der Properties in Abhängigkeit davon wie viele Literale identisch sind und an welchen Positionen der beiden Klauseln diese stehen. Durch diese Konstruktion wird erreicht, dass ein Tripel  $(s, p, o)$  mit den beiden Belegungsressourcen  $s$  und  $o$  äquivalent dazu ist, dass die Klauseln mit der Belegung, die den Ressourcen  $s$  und  $o$  entsprechen gleichzeitig erfüllbar sind. Wichtig hierbei ist, dass die Anzahl der Belegungsressourcen sowie der zwischen ihnen befindlichen Properties endlich ist. Dies folgt aus der Tatsache, dass es lediglich polynomiell viele unterschiedliche Klauseln der maximalen Länge 3 gibt, jede dieser Klauseln maximal  $2^3 - 1 \in \text{poly}(x)$  viele erfüllende Belegung besitzt und es nur 7 mögliche Properties für jedes Paar von Belegungsressourcen gibt. Insgesamt ist die Menge der Belegungsressourcen sowie Properties durch ein Polynom begrenzt und damit endlich.

Die Verweisressourcen dienen lediglich der Auflistung aller Möglichen Klauseln der maximalen Länge 3 von denen wir wissen, dass es maximal polynomiell viele gibt. Dieser Ressourcentyp besitzt nur eine einzige Art von Properties (set). Nämlich einen Verweis auf alle möglichen Belegungen in Form von Belegungsressourcen, die die Klausel erfüllen. Dies hat zur Folge, dass diese Art von Ressourcen und Ihre Properties ebenfalls einer endlichen Menge entsprechen.

Nachdem die fixierte und somit konstante Datenbank beschrieben ist, kann das vorgehen der transformierenden TM beschrieben werden. Für jede Kombination zweier Klausel  $k_i$  und  $k_j$  der KNF mit  $i, j \in [1..n]$  werden die folgenden Bedingung dem Body der Abfrage hinzugefügt:

$$\{ (k_i, \text{set}, ?X_i), (k_j, \text{set}, ?X_j), (?X_i, \text{satisfy}_{\text{Match}(i,j)}, ?X_j), (?X_j, \text{satisfy}_{\text{Match}(j,i)}, ?X_i) \}$$

Hierbei sind die  $k_x$  die Verweisressourcen, die den Klauseln entsprechen, während, die Variablen  $?X_k$  wegen der Datenbankdefinition nur von Belegungsressourcen instanziiert werden können. Die Funktion  $\text{Match}(k_1, k_2)$ , dient zudem der Bestimmung der Übereinstimmung von Variablen innerhalb der beiden Parameterklauseln und somit der Auswahl des richtigen Typs der Property  $\text{satisfy}_{\text{Typ}}$ .

Die Abfrage besitzt keine Einschränkungen, d.h.  $C \equiv \emptyset$  und der Header ist ein beliebiger konstanter Ausdruck. Aus der Definition folgt unmittelbar, dass die gegebene KNF lösbar ist wenn die Antwort nicht leer ist.

## 2. Daten Komplexität:

Da die Abfrage fixiert ist, ist die Größe  $|q|$ , d.h. die Anzahl der Tripeln des zu matchenden Graphen ebenfalls fixiert. Hieraus folgt, dass es nur polynomiell viele Möglichkeiten für einen zu matchenden Graphen gibt, genau genommen  $|D|^{|q|}$ , wobei  $|D|$  der Größe der Datenbank  $D$  entspricht. Da der Test jedes potentiellen Graphen in Konstantzeit gelingt, ist die Bewertung in dieser Version in polynomieller Zeit berechenbar.

Eine weitere Folge des obigen Beweises ist, dass die Größe des Antwortgraphen stets durch das Polynom  $|D|^{|q|}$  nach oben begrenzt wird. Daran ändert auch die Verwendung von Reification nichts, da dies die Datenbank nur um einen konstanten Faktor vergrößert.

## 6.2. Minimierung von Abfragen

Der *statischen Optimierung* von Abfragen muss eine große Bedeutung beigemessen werden, da wie aus Satz 3 ersichtlich wird, die Auswertung einer Abfrage eine exponentielle Laufzeit in der Abfragegröße aufweisen kann.

**Definition 14: Abfragehomomorphismus**

Unter einem *Homomorphismus*  $h : q_1 \rightarrow q_2$  für zwei Abfragen  $q_i \equiv (B_i, H_i, C_i) : i \in \{1, 2\}$  versteht man eine Ersetzung  $\Phi$  von Variablen und anonymen Ressourcen, so dass folgende Bedingungen gelten:

1.  $\Phi(H_1) = H_2$
2.  $\Phi(B_1) \subseteq B_2$
3.  $C_1 \subseteq C_2$

Des Weiteren gilt stets  $q_1 \subseteq q_2$ , wenn für alle Datenbanken  $D$  stets  $\text{ans}(q_1, D) \subseteq \text{ans}(q_2, D)$  erfüllt ist.

**Lemma 7:**

Für zwei Abfragen  $q_1, q_2$  gilt:

$$q_2 \subseteq q_1 \leftrightarrow \exists \text{ein Homomorphismus } h : q_1 \rightarrow q_2$$

**Beweis:**

$\rightarrow$ : Nach Annahme gilt  $q_2 \subseteq q_1$ , des Weiteren sei  $D$  die Datenbank, die aus  $B_2$  hervorgeht, wenn jede Variable  $?X$  in  $B_2$  durch die entsprechende Konstante  $c_{?X}$  ersetzt wird. Die Bewertung  $v$  sei hierzu die entsprechende Abbildung, d.h.  $v$  bildet jede Variable  $?X$  auf  $c_{?X}$  ab

$$\rightarrow v(H_2) \in \text{ans}(q_2, D) \subseteq \text{ans}(q_1, D)$$

$$\rightarrow \exists \text{ eine Bewertung } v' : D \models v'(B_1) \wedge v'(H_1) = v(H_2)$$

$\rightarrow$  Entspreche  $h$  der Ersetzung  $v^{-1} \circ v'$ , die Bedingung  $C_1 \subseteq C_2$  folgt aus  $q_2 \subseteq q_1$

$$\rightarrow \exists \text{ein Homomorphismus } h : q_1 \rightarrow q_2$$

$\leftarrow$ :  $\exists$  ein Homomorphismus  $h : q_1 \rightarrow q_2$

$$\rightarrow \forall \text{ Datenbanken } D \wedge \forall \text{ Bewertungen } v \text{ gilt: } D \models v(B_2) \rightarrow D \models v(\Phi(B_1))$$

$\rightarrow \forall v(H_2) \in \text{preans}(q_2, D)$  gilt  $v(H_1) \in \text{preans}(q_1, D)$  da  $\Phi(H_1) = H_2$  und  $C_1 \subseteq C_2$

$$\rightarrow q_2 \subseteq q_1$$

Eine Abfrage  $q = (H, B, C)$  wird *minimal* genannt, wenn keine andere Abfrage  $q' = (H', B', C')$  existiert, die äquivalent zu  $q$  ist und für die gleichzeitig  $|B'| < |B|$  gilt, wobei  $|B|$  die Anzahl der Elemente des Bodies bezeichnet.

**Lemma 8:**

Für jede Abfrage  $q = (H, B, C)$  existiert eine minimale Abfrage  $q_m = (H_m, B_m, C_m)$  die äquivalent zu  $q$  ist und für die  $B_m \subseteq B$  und  $C_m \subseteq C$  gilt.

**Beweis:**

Entweder ist  $q$  bereits die minimale Abfrage dann ist die obere Aussage trivialerweise erfüllt oder es existiert eine minimale Abfrage  $q'$ , die äquivalent zu  $q$  ist. Da  $q'$  eventuell die oberen Bedingungen für den Body und die Einschränkungen nicht erfüllt, muss sie zu der gesuchten Abfrage  $q_m$  transformiert werden. Man beachte, dass wegen  $q \equiv q'$  ebenfalls  $q' \subseteq q$  sowie  $q \subseteq q'$  gilt. Deshalb existieren die beiden folgenden Homomorphismen  $\Phi_1 : q \rightarrow q'$  sowie  $\Phi_2 : q' \rightarrow q$ . Die Hintereinanderausführung dieser führt dann zur gesuchten minimalen Abfrage, da stets  $q_m = \Phi_2 \circ \Phi_1(q)$  gilt.

**Bemerkung 7:**

Man muss sich stets vor Augen halten, dass die Minimierung einer Abfrage nicht mit der Umwandlung des Bodies einer Abfrage zu einem äquivalenten kompakten Graphen übereinstimmt. Der Grund hierfür ist, dass der Homomorphismus zur Minimierung der Abfrage zusätzlich den Header der Abfrage unverändert lassen muss.

**Lemma 9:**

Für zwei Abfragen  $q_1, q_2$  sind die folgenden Probleme NP-vollständig:

1. Gilt  $q_1 \equiv q_2$ ?
2. Gilt  $q_1 \subseteq q_2$ ?

**Beweis:**

1. Für den Nachweis, dass das Problem NP-hart ist verweise ich auf die Behauptung in [4], dass das Äquivalenzproblem für zwei Tableaus ebenfalls NP-hart ist. Da für die Äquivalenz zweier Abfragen stets  $C_1 = C_2$  gelten muss, lassen sich diese zudem problemlos als klassisches Tableau mit drei Attributen kodieren.
2. Da sich  $q_1 \equiv q_2$  auf die Lösung der zwei Teilproblem Probleme  $q_1 \subseteq q_2$  sowie  $q_2 \subseteq q_1$  reduzieren lässt, folgt dass diese Problem NP-hart ist unmittelbar aus der NP-Vollständigkeit des Problems  $q_1 \equiv q_2$ .

Die Zugehörigkeit beider Probleme zu NP folgt unmittelbar, aus der Tatsache, dass ein Homomorphismus ein polynomiell verifizierbarer Zeuge ist.

**6.3. Eliminierung von Redundanzen**

Wie bereits die Bemerkung 7 gezeigt hat, weisen Antworten typischer Weise *Redundanzen* auf, die vermieden werden sollten. Ideal wäre es wenn die Antwortmenge  $\text{ans}(q, D)$  einem kompakten Graphen entsprechen würde und somit frei von Redundanz wäre. Die folgende Betrachtung verfolgt dieses Ziel und untersucht dabei die auftretende Komplexität.

Der triviale Ansatz eine *redundanzfreie Antwort* zu erhalten, besteht zunächst aus der Bestimmung des Graphen  $G = \text{ans}(q, D)$  und der folgenden Berechnung eines kompakten Graphen der äquivalent zu  $G$  ist. Wie wir bereits gesehen haben ist die letzte Berechnung schwierig, jedoch existiert für den Worst-Case keine bessere Alternative, wie der folgende Satz beweist.

**Satz 4:**

Die Beantwortung der Frage, ob für eine beliebige Datenbank  $D$  und Anfrage  $q$   $\text{ans}_u(q, D)$  kompakt ist, ist NP-Vollständig (in der Größe von  $D$ ).

**Beweis:**

Dieser Satz ist eine unmittelbare Konsequenz aus der Reduktion des Problems aus Satz 2 auf dieses Problem, wobei die Reduktion ausnutzt, dass für die Union-Semantik eine datenunabhängige Identitätsabfrage existiert (Beispiel 6).

Bei der Verwendung der Merge-Semantik kann die obige Frage unter bestimmten Vorbedingungen effizient, d.h. bereits in polynomieller Zeit beantwortet werden, wie wir gleich sehen werden.

**Satz 5:**

Für eine beliebige kompakte Datenbank  $D$  und eine beliebige Abfrage  $q$  mit kompakten Header, kann die Frage, ob  $\text{ans}_m(q, D)$  kompakt ist, bereits in polynomieller Zeit gelöst werden.

**Beweis:**

Sei  $M = \text{ans}_m(q, D)$ , des Weiteren sollen mit dem Begriff einfaches Mapping solche Mappings bezeichnet werden, die Abbildungen von einfachen Antworten nach  $M$  entsprechen.

Die Hauptidee ist, dass bei der Verwendung von der Merge-Semantik die einfachen Antworten keine Variablen teilen und somit alle Mappings  $\mu : M \rightarrow M$  genau der Vereinigung von einfachen Mappings  $\mu_j : H_j \rightarrow M$  für jede einfache Antwort  $H_j$  entsprechen. Deshalb muss ein Algorithmus der ein Mapping  $\mu : M \rightarrow M$  sucht, das zu einer echten Instanz führt lediglich einfache Mappings betrachten, wobei zwei Tests durchzuführen sind. Der erste ist, die Frage ob zumindest ein Mapping besteht, das zu einer Instanz eines anderen führt und der zweite ist, die Frage, ob es zwei Mappings gibt, die sich lediglich in ihren anonymen Ressourcen unterscheiden.

Da beide Tests in  $\text{poly}(\# \text{ einfache Mappings})$  durchführbar sind, folgt dass sie in  $\text{poly}(|D|)$  realisiert werden können, da wir bereits im zweiten Teil des Beweises von Satz 3 gesehen haben, dass die Anzahl der einfachen Mappings durch  $\text{poly}(|D|)$  begrenzt sind.

## 7. Schlussfolgerungen und RAL

Die angestellten Betrachtungen haben gezeigt, dass mit der Verwendung von *RDF Datenbanken* neuen Anforderungen an Abfragesprachen einhergehen. Dies ist eine unmittelbare Folge von Konstrukten wie den *anonymen Ressourcen* und *Reification*, die dem RDF eine enorme Ausdruckskraft bieten, ohne dabei die Komplexität der Abfragen erheblich zu erhöhen.

Dabei spielen vor allem die anonymen Ressourcen eine entscheidende Rolle, da wie wir gesehen haben durch die existierenden Interpretationsmöglichkeiten zwei unterschiedliche *Semantiken* entstehen, die zu verschiedenen Anforderungen an die Bestimmung von Anfrageergebnissen führen. Dies zeigt, dass die theoretischen Aspekte, auf denen RDF Abfragesprachen basieren, noch weiter untersucht werden müssen.

Insbesondere das Fehlen von einer *Algebra*, die die Definition sowie den Vergleich von unterschiedlichen RDF Abfragesprachen ermöglicht, zeigt, dass auf diesem Gebiet noch Forschungsbedarf existiert, obwohl mit der Definition von RAL in [5], eine Algebra für RDF Abfragesprachen vorgeschlagen wird, die einen ersten Schritt in diese Richtung unternimmt. Jedoch ist dies erst der Anfang, der aber bereits ein Hauptproblem von RDF Abfragesprachen aufzeigt. Nämlich dass sie (die hier vorgestellte Abfragesprache ist davon nicht ausgenommen) nur einer bloßen Definition der Logik für die Definition von Abfragen entsprechen und die interne Arbeitsweise hinter API-Aufrufen verbergen, so dass keine einheitlicher Standard existiert.

Darüber Hinaus zeigt RAL, dass man sich nicht nur auf Operatoren für die Extraktion der Daten beschränken darf, sondern aufgrund der RDF zugrunde liegenden Struktur ebenfalls Operatoren für Wiederholungen sowie für die Konstruktion von RDF Datenbanken benötigt.

## 8. Literaturverzeichnis

- [1] Manola F., Miller E.: *RDF Primer*. W3C Working Draft 10 October 2003  
[www.w3.org/TR/2003/WD-rdf-primer-20031010/](http://www.w3.org/TR/2003/WD-rdf-primer-20031010/)
- [2] Hayes P.: *RDF Semantics*. W3C Working Draft 10 October 2003  
[www.w3.org/TR/2003/WD-rdf-mt-20031010/](http://www.w3.org/TR/2003/WD-rdf-mt-20031010/)
- [3] Tolle K.: *Analyzing and Parsing RDF*. Diplomarbeit, Universität Hannover, Fachbereich Mathematik, Studienrichtung Informatik, Januar 2000  
<http://www.kbs.uni-hannover.de/Arbeiten/Diplomarbeiten/00/tolle/AuPRDF.ps>
- [4] Gutierrez C., Hurtado C., Mendelzon A.: *Formal aspects of querying RDF databases*  
[www.cs.uic.edu/~ifc/SWDB/papers/Gutierrez\\_etal.pdf](http://www.cs.uic.edu/~ifc/SWDB/papers/Gutierrez_etal.pdf)
- [5] Barna P., Frasinca F., Houben G., Vdovjak R.: *RAL: an Algebra for Querying RDF*  
[wwwis.win.tue.nl/~hera/papers/WISE2002/ral.pdf](http://wwwis.win.tue.nl/~hera/papers/WISE2002/ral.pdf)
- [6] Hell P., Nestril J.: *The core of a graph*. In: *Discrete Mathematics* 109. North-Holland 1992, S. 117 – 126
- [7] Ullman J. D.: *Principles of DATABASE SYSTEMS*. 2nd ed. Rockville: Computer Science Press, 1984. – ISBN 0-7167-8069-0