Semantic Web Grundlagen

Lösung zur Übung 6: SPARQL & Entailment

Birte Glimm

WS 2011/2012

Lösung (6.1)**.**
Wir beginnen mit dem Abfragemuster, welches ein `GroupGraphPattern` Element ist. Das Abfragemuster selbst enthält eine alternative Abfrage, besteht also aus einem `GroupOrUnionGraphPattern` Element der Form

`GroupGraphPattern` UNION `GroupGraphPattern`

ist (siehe Folie 39 der SPARQL Semantik Vorlesung, Vorlesung 15). Wir starten daher mit der Übersetzung entsprechend Algorithmus 1 (Folie 40) und darin entsprechend des Falls für `GroupGraphPattern`, also mit Algorithmus 4 (Folie 43). Hier trifft das letzte else statement zu (Zeile 12), daher erhalten wir:

$$\mathsf{Join}(\mathsf{Z},\ \mathsf{algbr}(\mathsf{G}))$$

mit G dem `GroupOrUnionGraphPattern`. Algorithmus 1 verweist entsprechend auf Algorithmus 2 (Folie 41). Wir haben zwei Elemente, die übersetzt werden müssen, um die resultierenden Algebra Objekte mit der Union Funktion zu verbinden. Wir haben also:

$$\mathsf{Join}(\mathsf{Z},\ \mathsf{Union}(\mathsf{algbr}(\mathsf{G}_1),\ \mathsf{algbr}(\mathsf{G}_2)))$$

wobei G_1 und G_2 die beiden Gruppenelemente sind, die über `UNION` verbunden sind.

Da sowohl G_1 als auch G_2 `GroupGraphPattern` Elemente sind, gehen wir, entsprechend Algorithmus 1, zu Algorithmus 4. Da G_1 einen Filter hat, erhalten wir einmal die Übersezung für den `TriplesBlock`, über den dann der Filter angewendet wird:

Filter(regex(`?id`, `"markus.kr*"`),
Join(Z, Bgp(`?x foaf:icqChatID ?id . ?x foaf:name ?name`))).

Für G_2 verwenden wir wieder Algorithmus 4 und darin das letzte else statement (Zeile 12), wobei wir dann in Algorithmus 1 wieder den Fall für `TriplesBlock` verwenden:

$$\mathsf{Join}(\mathsf{Z},\ \mathsf{Bgp}(\texttt{?x foaf:name ?name . ?x foaf:icqChatID "bglimm"}))$$

Wenn wir nun alles zusammensetzen, erhalten wir:

Join(Z, Union(Filter(regex(`?id`, `"markus.kr*"`),
Join(Z, Bgp(`?x foaf:icqChatID ?id . ?x foaf:name ?name`))),
Join(Z, Bgp(`?x foaf:name ?name . ?x foaf:icqChatID "bglimm"`))))

Wir können nun vereinfachen und erhalten:

Union(Filter(regex(?id, "markus.kr*"),
Bgp(?x foaf:icqChatID ?id . ?x foaf:name ?name)),
Bgp(?x foaf:name ?name . ?x foaf:icqChatID "bglimm"))

Sei E das bisherige Algebra Objekt. Wir können nun noch die restlichen Abfrage Elemente übersetzen (siehe Folie 47) und erhalten:

$$\mathsf{Project}(\mathsf{ToList}(\mathsf{E}), \{?\texttt{name}\})$$

Lösung (6.2)**.**
Wir listen zuerst Tripel die unter der RDFS Semantik folgen und für die Abfrage relevant sind. Die Tripel folgen entsprechend der RDFS Regeln (Vorlesung 6, Folien 41–43). Die relevanten Regeln und Tripel werden jeweils in der linken Spalte angezeigt.

rdfs9 + (1) + (11) → (14) w3c:sparql11-entailment rdf:type ex:Publikation .
rdfs9 + (6) + (11) → (15) w3c:rdf-sparql-query rdf:type ex:Publikation .
rdfs7 + (7) + (13) → (15) w3c:rdf-sparql-query ex:autoren _:l2 .

Wenn wir alle Tripel materialisieren wollten, die unter RDFS folgen, müssten wir noch diverse weitere Tripel hinzufügen. Wir beschränken uns hier auf die für die Antwort relevanten Tripel. Obwohl in diesem Fall keine frischen leeren Knoten durch die RDFS Regeln eingführt wurden, sollten Sie sich darüber klar sein, dass durch die Regeln erzeugte leere Knoten nicht in Anfrageantworten zurückgeliefert werden. Wir erhalten die folgenden Lösungen durch die Auswertung des BGPs:

	pub	**seq**	**ind**	**aut**
μ_1 :	w3c:sparql11-entailment	_:l1	rdf:type	rdf:Seq
μ_2 :	w3c:sparql11-entailment	_:l1	rdf:_1	"Birte Glimm"
μ_3 :	w3c:sparql11-entailment	_:l1	rdf:_2	"Chimezie Ogbuji"
μ_4 :	w3c:rdf-sparql-query	_:l2	rdf:type	rdf:Seq
μ_5 :	w3c:rdf-sparql-query	_:l2	rdf:_1	"Andy Seaborne"
μ_6 :	w3c:rdf-sparql-query	_:l2	rdf:_2	"Eric Prud'hommeaux"

Die Projektion zu berechnen ist dann einfach.

Lösung (6.3)**.**
Eine Möglichkeit wäre einen Filter auf ?aut anzuwenden, so dass nur Literale als Bindungen erlaubt werden:

```
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  ex: <http://example.org/>
SELECT  ?aut ?pub
WHERE   { ?pub rdf:type ex:Publikation .
          ?pub ex:autoren ?seq .
          ?seq ?ind ?aut
          FILTER ISLITERAL(?aut) }
```

Andere Lösungen mit anderen Filtern sind ebenfalls möglich.

Eine noch bessere Lösung nutzt die Semantik von RDFS auf und insbesondere, dass IRIs der Form `rdf:_n` den Typ `rdfs:ContainerMembershipProperty` haben:

```
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  ex: <http://example.org/>
SELECT  ?aut ?pub
WHERE   { ?pub rdf:type ex:Publikation .
          ?pub ex:autoren ?seq .
          ?seq ?ind ?aut
          ?ind rdf:type rdfs:ContainerMembershipProperty . }
```

Lösung (6.4)**.**
Die folgenden Tripel folgen unter RDFS Semantik und sind relevant für die Anfrage.

$$\begin{aligned} & (1) && \texttt{w3c:sparql11-entailment rdf:type ex:WorkingDraft .} \\ \text{rdfs4a} + (1) \rightarrow\ & (14) && \texttt{w3c:sparql11-entailment rdf:type rdfs:Resource .} \\ \text{rdfs9} + (1) + (11) \rightarrow\ & (15) && \texttt{w3c:sparql11-entailment rdf:type ex:Publikation .} \end{aligned}$$

Dementsprechend hat die Anfrage drei Antworten. Konsequenz (14) scheint vielleicht überraschend, aber unter RDFS können wir diverse derartige Tripel ableiten. Wenn solche Tripel nicht in den Antworten gewünscht sind, kann ein Filter verwendet werden, um diese Antworten rauszufiltern.

Lösung (6.5)**.**
Das Tripel

$$\texttt{w3c:sparql11-entailment ex:autoren \_:x}$$

folgt unter RDFS Semantik, da die leeren Knoten wie existentielle Variablen interpretiert werden und Tripel (2) belegt, dass `w3c:sparql11-entailment` mit der Property `ex:autoren` zu einem (*some*) Element verbunden ist (dem leeren Knoten `_:l1`, wobei das Label des leeren Knoten unerheblich ist).

Für eine Boolsche Abfrage (in diesem Fall ohne echte Variablen, aber mit einem leeren Knoten) haben wir zwei Möglichkeiten: es gibt eine Lösung mit einer Funktion μ wobei μ eine leere Domäne hat (keine echten Variablen) oder es gibt keine Lösung. Im ersten Fall ist die Antwort `true` (wahr) und im zweiten Fall `false` (falsch).

Für das RDFS Entailment Regime, arbeiten wir mit einer Skolem Funktion, die leere Knoten des Graphen auf frische Konstanten abbildet, also Konstanten, die vorher nicht in dem Graphen oder der Abfrage vorkamen. Nehmen wir an, dass `_:l1` auf `skol:l1` abgebildet wird, also $\mathsf{sk}(\texttt{\_:l1}) = \texttt{skol:l1}$. Da das Abfragemuster einen leeren Knoten enthält, müssen wir ein passendes RDF Instanz Mapping finden. Dieses muss so sein, dass die Anwendung auf das Abfragemuster mit anschliessender Skolemisierung zu Tripeln führt, die ground/grundiert, also variablenfrei, sind und unter RDFS folgen. Sei μ eine Funktion mit leerer Domäne und σ: `_:x` $\mapsto$ `_:l1`, dann gilt

$$\begin{aligned} & \mathsf{sk}(\mu(\sigma(\texttt{w3c:sparql11-entailment ex:autoren \_:x}))) \\ = \ & \texttt{w3c:sparql11-entailment ex:autoren skol:l1.} \end{aligned}$$

Dieses Tripel ist ground/grundiert und folgt unter RDFS (sogar einfach) aus der Skolemisierung des abgefragten Graphen $\mathsf{sk}(\mathsf{G})$. Daher ist die Antwort `true` (wahr).

Lösung (6.6)**.**
Eine SPARQL Abfrage kann nicht ohne weiteres zwischen direkten und indirekten Unterklassen unterscheiden. Mit einem kleinen Trick und SPARQL 1.1 Features, lässt sich das Problem aber lösen. Wir können die folgende Abfrage verwenden:

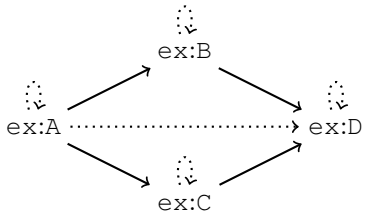
```
PREFIX    rdfs: <http://http://www.w3.org/2000/01/rdf-schema#>
SELECT    ?sub ?sup
WHERE     { ?sub rdfs:subClassOf ?x . ?x rdfs:subClassOf ?sup }
GROUP BY  ?sub ?sup
HAVING    COUNT(*) = 2
```

Paare aus Unterklasse und direkter Oberklasse zeichnen sich dadurch aus, dass es genau zwei Varianten von Bindungen in denen einmal `?x` = `?sub` und einmal `?x` = `?sup` gilt. Paare bei denen die Oberklasse indirekt ist, erlauben mehr als zwei Möglichkeiten um `?sub` und `?sup` über `rdfs:subClassOf` zu verbinden.

Als Beispiel, nehmen wir folgenden einfachen RDF Graphen:

```
@prefix rdfs: <http://http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .
ex:A rdfs:subClassOf ex:B .
ex:A rdfs:subClassOf ex:C .
ex:B rdfs:subClassOf ex:D .
ex:C rdfs:subClassOf ex:D .
```

Visualisiert durch den folgenden Graphen, in dem Kanten explizit gegebene `rdfs:subClassOf` Beziehungen darstellen.



Die Auswertung des BGPs ergibt die folgenden Bindungen:

	sub	x	sup
μ_1 :	ex:A	ex:A	ex:A
μ_2 :	ex:A	ex:A	ex:B
μ_3 :	ex:A	ex:A	ex:C
μ_4 :	ex:A	ex:A	ex:D
μ_5 :	ex:A	ex:B	ex:B
μ_6 :	ex:A	ex:B	ex:D
μ_7 :	ex:A	ex:C	ex:C
μ_8 :	ex:A	ex:C	ex:D
μ_9 :	ex:A	ex:D	ex:D
μ_{10} :	ex:B	ex:B	ex:B
μ_{11} :	ex:B	ex:B	ex:D
μ_{12} :	ex:B	ex:D	ex:D
μ_{13} :	ex:C	ex:C	ex:C
μ_{14} :	ex:C	ex:C	ex:D
μ_{15} :	ex:C	ex:D	ex:D
μ_{16} :	ex:D	ex:D	ex:D

Gruppierung nach `?sub ?sup` ergibt neun Gruppen:

	sub	x	sup
μ_1 :	ex:A	ex:A	ex:A
μ_2 :	ex:A	ex:A	ex:B
μ_5 :	ex:A	ex:B	ex:B
μ_3 :	ex:A	ex:A	ex:C
μ_7 :	ex:A	ex:C	ex:C
μ_4 :	ex:A	ex:A	ex:D
μ_6 :	ex:A	ex:B	ex:D
μ_8 :	ex:A	ex:C	ex:D
μ_9 :	ex:A	ex:D	ex:D
μ_{10} :	ex:B	ex:B	ex:B
μ_{11} :	ex:B	ex:B	ex:D
μ_{12} :	ex:B	ex:D	ex:D
μ_{13} :	ex:C	ex:C	ex:C
μ_{14} :	ex:C	ex:C	ex:D
μ_{15} :	ex:C	ex:D	ex:D
μ_{16} :	ex:D	ex:D	ex:D

Die Gruppen mit Kardinalität 1 sind die reflexiven Beziehung einer Klasse mit sich selbst. Bei den direkten Unterklassenbeziehungen gibt es jeweils zwei Elemente in der Gruppe: einmal mit `?x` gebunden and die Unterklasse und einmal mit `?x` gebunden an die Oberklasse. Zwischen `ex:A` und `ex:D` gibt es auch indirekte `rdfs:subClassOf` Beziehungen, die zu einer erhöhten Kardinalität der Gruppe führen.

Das `COUNT(*)` Kriterium in der `HAVING` Bedingung, filtert nun alle Gruppen mit Kardinalität ungleich zu zwei raus:

	sub	x	sup
μ_2 :	ex:A	ex:A	ex:B
μ_5 :	ex:A	ex:B	ex:B
μ_3 :	ex:A	ex:A	ex:C
μ_7 :	ex:A	ex:C	ex:C
μ_{11} :	ex:B	ex:B	ex:D
μ_{12} :	ex:B	ex:D	ex:D
μ_{14} :	ex:C	ex:C	ex:D
μ_{15} :	ex:C	ex:D	ex:D

SPARQL wird automatisch die gruppierten und selektierten Variablen sampeln, also nur einen Wert pro Gruppe zurückgeben. Da die selektierten Werte auch zur Gruppierung verwendet wurden, ist klar, dass jeder innerhalb der Gruppe gleich ist. `SAMPLE` ist also eindeutig. Wir erhalten also wie gewünscht:

sub	sup
ex:A	ex:B
ex:A	ex:C
ex:B	ex:D
ex:C	ex:D

Die Lösung funktioniert noch nicht wie gewünscht, wenn es äquivalente Klassen gibt. Diese würden dann ebenfalls an `?x` binden und somit zu mehr als zwei Ergebnissen in der Gruppe führen.