Semantische Technologien für das Web

Überblick

1. RDF Standardisierung
2. RDF Anfragesprache: SPARQL
3. RDFS: RDF Schema
4. Abbildung relationaler Datenbanken nach RDF

Literatur: W3C-Standards.

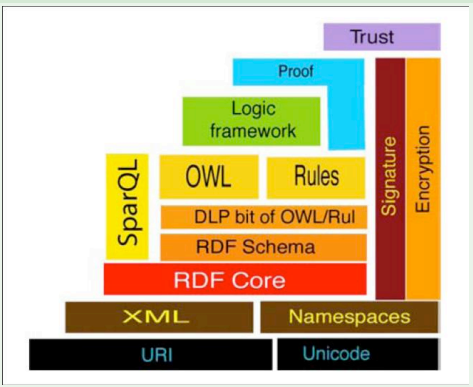
Warum semantische Technologien für das Web?

Das Web:

- Daten übertragen: HTTP
- Daten Adressieren: URI (Uniform Resource Identifier)
- Daten darstellen: HTML
- Daten austauschen: XML
- Daten verstehen: ???

Ontologien

- *Formal, explicit specification of a shared conceptualization of a domain.*
- Grundlage des *Semantic Web*.



Resource Description Framework: RDF

Internet Engineering Task Force RFC2396: Resource, Uniform Resource Identifier (URI)

A **Uniform Resource Identifier (URI)** is a compact string of characters for identifying an abstract or physical resource.

. . .

A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources.

RDF: W3C Recommendation 10 February 2004

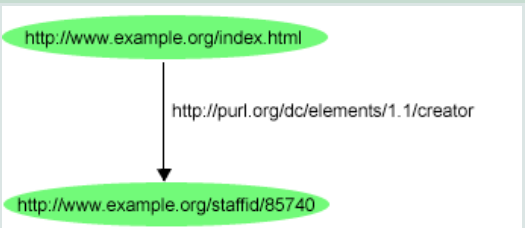
Simple data model to express statements about the world, designed to represent information in a minimally constraining, flexible way: *subject* – *predicate* – *object*-triples define a labelled directed graph.

- Subject: Resource or blank node, i.e. node that is neither a URI nor literal.
- Predicate: Property.
- Object: Resource, literal or blank node.

Extensible URI-based vocabulary; a URI is an identifier, not a location on the web.

Graph representation of a RDF statement

Quelle: http://www.w3.org/TR/rdf-primer/

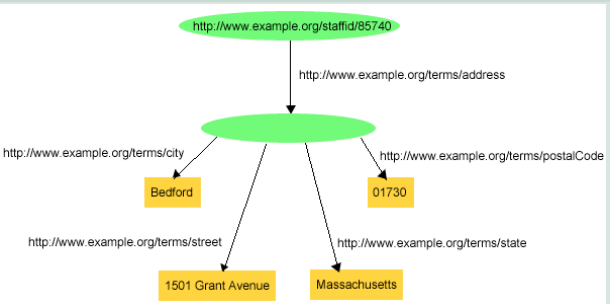


RDF

- URI, blank node, literal.
- Literals may be plain or typed.
  - A *plain literal* is a string with an optional language tag (self-denoting). Example: *adda-adda@alemannisch*.
  - A *typed literal* is a string combined with a datatype URI, which denotes the member of the identified datatype's value space obtained by applying the lexical-to-value mapping to the literal string. Example: *"1"^^xsd:integer*.
- XML-based syntax. Supporting use of XML schema datatypes.
- Anyone might make statements about anything.
- Formal semantics and provable inference.

RDF-graph with blank nodes and triple notation

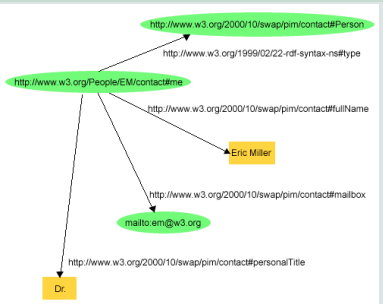
Quelle: http://www.w3.org/TR/rdf-concepts/



http://www.expl.org/staffid/85740	http://www.expl.org/terms/address	_:1
_:1	http://www.expl.org/terms/city	"Bedfort"
_:1	http://www.expl.org/terms/street	"1501 Grant Avenue"
_:1	http://www.expl.org/terms/state	"Massachusetts"
_:1	http://www.expl.org/terms/postalCode	"01730"

RDF-graph with fragment identifiers and XML serialization

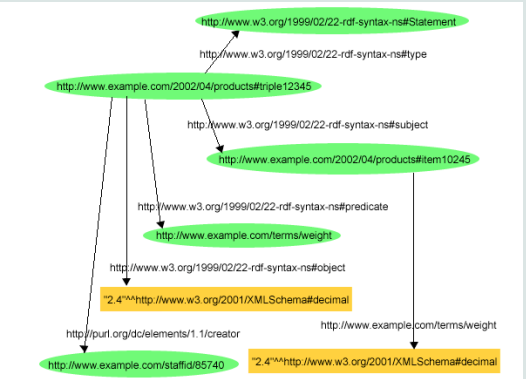
Quelle: http://www.w3.org/TR/rdf-primer/




```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

Statements about statements: Reification

Quelle: http://www.w3.org/TR/rdf-primer/



Ein RDF Statement kann selbst Resource sein. Die Komponenten eines Statements werden durch Prädikate *subject*, *predicate*, *object* identifiziert.

Predefined resources and properties: RDF vocabulary

- Classes

  rdf:Property, *rdf:Statement*, *rdf:XMLLiteral*, *rdf:Seq*, *rdf:Bag*, *rdf:Alt*, *rdf:List*
- Poperties

  rdf:type, *rdf:subject*, *rdf:predicate*, *rdf:object*, *rdf:first*, *rdf:rest*, *rdf:_n*, *rdf:value*
- Resources

  rdf:nil

RDF Containers and Collections

- Types of containers:
  - *Bag*: unordered set of items
  - *Seq*: ordered set of items
  - *Alt*: set of alternatives
- Collection type: *List*

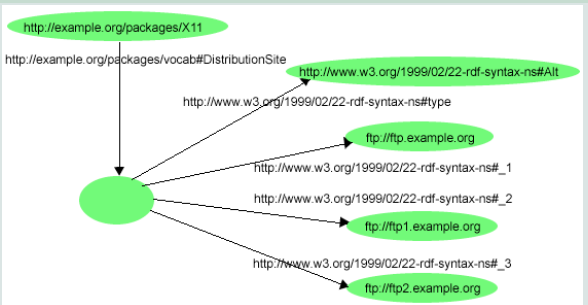
  Collections differ from containers in allowing branching structure and in having an explicit terminator.
- No RDF-defined semantics; this is up to the application.

RDF Container: Bag

Quelle: http://www.w3.org/TR/rdf-primer/

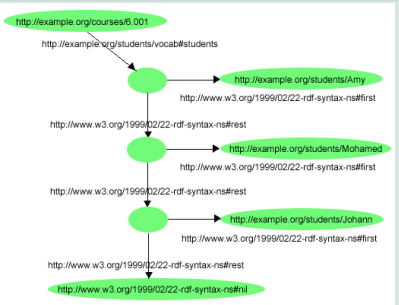
RDF Container: Alt

Quelle: http://www.w3.org/TR/rdf-primer/



RDF Collection: List

Quelle: http://www.w3.org/TR/rdf-primer/



RDF Anfragesprache: SPARQL

SPARQL queries

- PREFIX: mechanism for abbreviating URIs
- SELECT: identifies the variables to be returned in the query answer
- FROM: names the graph to be queried
- WHERE: query pattern

Outline:

- First impressions by examples.
- Semantics.
- More examples to complete the picture.

Quelle:
W3C Recommendation January 15th, 2008

Example 1 (Writing a Simple Query):

Data

```
<http://example.org/book/book1>
      <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Query

```
SELECT ?title
WHERE {
      <http://example.org/book/book1>
            <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Example 2 (Multiple Matches):

Data

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a foaf:name   "Johnny Lee Outlaw" .
_:a foaf:mbox   <mailto:jlow@example.com> .
_:b foaf:name   "Peter Goodguy" .
_:b foaf:mbox   <mailto:peter@example.org> .
_:c foaf:mbox   <mailto:carol@example.org> .
```

Query

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
 { ?x foaf:name ?name .
   ?x foaf:mbox ?mbox }
```

Result

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Example 3 (Blank Node Labels in Query Results):

Data

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name   "Alice" .
_:b  foaf:name   "Bob" .
```

Queries

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

SELECT ?x ?name
WHERE  { ?x foaf:name ?name }
```

Result

x	name
_:c	"Alice"
_:d	"Bob"

Example 4 (Graph Patterns):

version 1: basic

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  {
         ?x foaf:name ?name .
         ?x foaf:mbox ?mbox .
       }
```

version 2: group

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { { ?x foaf:name ?name . }
         { ?x foaf:mbox ?mbox . }
       }
```

Example 5 (Filter):

version 1

```
{  ?x foaf:name ?name .
   ?x foaf:mbox ?mbox .
   FILTER (?name = "Smith")
}
```

version 2

```
 {  FILTER (?name = "Smith")
   ?x foaf:name ?name .
   ?x foaf:mbox ?mbox .
 }
```

version 3

```
{  ?x foaf:name ?name .
   FILTER (?name = "Smith")
   ?x foaf:mbox ?mbox .
}
```

Operators in FILTER

```
not !
bound
isIRI, isBlank, isLiteral
str
lang
datatype
logical-or ||, logical-and &&
RDFterm-equal =, sameTerm
langMatches
regex
```

Example 6 (Optional (left-associative)):

Data

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

_:a  rdf:type        foaf:Person .
_:a  foaf:name       "Alice" .
_:a  foaf:mbox       <mailto:alice@example.com> .
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  rdf:type        foaf:Person .
_:b  foaf:name       "Bob" .
```

Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x  foaf:mbox  ?mbox }
       }
```

Result

```
  name                mbox
"Alice"     <mailto:alice@example.com>
"Alice"     <mailto:alice@work.example>
"Bob"
```

Result

```
  name                mbox
"Alice"     <mailto:alice@example.com>
"Alice"     <mailto:alice@work.example>
"Bob"
```

Example 7 (Constraints in Optional Pattern Matching):

Data

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

Query

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>

SELECT  ?title ?price
WHERE   { ?x dc:title ?title .
          OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) }
        }
```

Result

```
    title            price
"SPARQL Tutorial"
"The Semantic Web"     23
```

Example 8a (Multiple Optional Graph Patterns):

Data

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice" .
_:a  foaf:homepage <http://work.example.org/alice/> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox  <mailto:bob@work.example> .
```

Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x foaf:mbox ?mbox } .
         OPTIONAL { ?x foaf:homepage ?hpage }
       }
```

Result

```
  name              mbox                hpage
"Alice"                          <http://work.example.org/alice/>
"Bob"       <mailto:bob@work.example>
```

Example 8b (Multiple Optional Graph Patterns):

Data

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice" .
_:a  foaf:homepage <http://work.example.org/alice/> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox  <mailto:bob@work.example> .
```

Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x foaf:mbox ?mbox .
                    OPTIONAL { ?x foaf:homepage ?hpage }
                  }
       }
```

Result

```
  name              mbox                  hpage

"Bob"     <mailto:bob@work.example>
"Alice"
```

Example 9a (Matching Alternatives):

Data

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .

_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .
_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .
_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
```

Query

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?title
WHERE  { { ?book dc10:title  ?title } UNION
         { ?book dc11:title  ?title }
       }
```

Result

```
      title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"
```

Example 9b (Matching Alternatives):

Data

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .

_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .
_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .
_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
```

Query

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?title
WHERE  { { ?book dc10:title ?title .  ?book dc10:creator ?author }
         UNION
           { ?book dc11:title ?title .  ?book dc11:creator ?author }
       }
```

Result

```
author              title
"Bob"     "SPARQL Protocol Tutorial"
"Alice"   "SPARQL Query Language Tutorial"
```

Example 11a (bound):

Data

```
@prefix foaf:        <http://xmlns.com/foaf/0.1/> .
@prefix dc:          <http://purl.org/dc/elements/1.1/> .
@prefix xsd:         <http://www.w3.org/2001/XMLSchema#> .

_:a  foaf:givenName  "Alice".

_:b  foaf:givenName  "Bob" .
_:b  dc:date"2005-04-04T04:04:04Z"^^xsd:dateTime .
```

Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?name
WHERE { ?x foaf:givenName  ?givenName .
        OPTIONAL { ?x dc:date ?date } .
        FILTER ( bound(?date) ) }
```

Result

```
givenName
  "Bob"
```

Example 11b (!bound):

Data

```
@prefix foaf:        <http://xmlns.com/foaf/0.1/> .
@prefix dc:          <http://purl.org/dc/elements/1.1/> .
@prefix xsd:         <http://www.w3.org/2001/XMLSchema#> .

_:a  foaf:givenName  "Alice".

_:b  foaf:givenName  "Bob" .
_:b  dc:date"2005-04-04T04:04:04Z"^^xsd:dateTime .
```

Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
SELECT ?name
WHERE { ?x foaf:givenName  ?name .
        OPTIONAL { ?x dc:date ?date } .
        FILTER (!bound(?date)) }
```

Result

```
givenName
  "Alice"
```

Result

```
One may test that a graph pattern is not expressed by specifying
an OPTIONAL graph pattern that introduces a variable and testing
to see that the variable is not bound.

This is called Negation as Failure in logic programming.
```

Semantik von SPARQL?

The outcome of executing a SPARQL query is defined by a series of steps, starting from the SPARQL query as a string, turning that string into an abstract syntax form, then turning the abstract syntax into a SPARQL abstract query comprising operators from the SPARQL algebra.
This abstract query is then evaluated on an RDF dataset.

Semantics of SPARQL

Algebra operators

$\bowtie, \cup, \setminus$, left outerjoin $⟕$

First simplify syntax using binary operators: UNION, AND, OPT, FILTER

instead of

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
SELECT ?name
WHERE { ?x foaf:name  ?name .
        ?x foaf:givenName  ?name .
        OPTIONAL { ?x dc:date ?date } .
        FILTER (!bound(?date)) }
```

now

```
((
  ((?x,foaf:name,?name) AND (?x,foaf:givenName,?name)
  ) OPT (?x,dc:date,?date)
 ) FILTER !bound(?date)
)
```

Jorge Pérez, Marcelo Arenas, Claudio Gutierrez: Semantics and Complexity of SPARQL. International Semantic Web Conference 2006: 30-43

Assume pairwise disjoint infinite sets I, B, and L (IRIs, Blank nodes, and literals).

- A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple.
- IL is the union $I \cup L$, and T the union $I \cup B \cup L$.
- V is an infinite set of variables disjoint from the other sets.

Definitions 1

A SPARQL graph pattern expression is defined recursively as follows:

(1) A tuple from $(T \cup V) \times (I \cup V) \times (T \cup V)$ is a graph pattern (a triple pattern).

(2) If P1 and P2 are graph patterns, then expressions (P1 AND P2), (P1 OPT P2), and (P1 UNION P2) are graph patterns.

(3) If P is a graph pattern and R is a SPARQL built-in condition, then the expression (P FILTER R) is a graph pattern.

A SPARQL built-in condition is constructed using elements of the set $V \cup T$ and constants, logical connectives $\neg, \wedge, \vee$, inequality symbols $<, \leq, \geq, >$ the equality symbol $=$, unary predicates like bound, isBlank, and isIRI, etc.

Definitions 2

- A mapping μ from V to T is a partial function $\mu : V \rightarrow T$. For a triple pattern t we denote by $\mu(t)$ the triple obtained by replacing the variables in t according to μ.
- The domain of μ, $dom(\mu)$, is the subset of V where μ is defined.
- Two mappings μ_1 and μ_2 are compatible when for all $x \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping.

  Two mappings with disjoint domains are always compatible.

  The empty mapping (i.e. the mapping with empty domain) μ_0 is compatible with any other mapping.

Definitions 3

- Let Ω_1 and Ω_2 be sets of mappings.
- $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ are compatible mappings } \}$,
- $\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}$,
- $\Omega_1 \setminus \Omega_2 = \{\mu \in \Omega_1 \mid \text{for all } \mu' \in \Omega_2,\ \mu \text{ and } \mu' \text{ are not compatible}\}$,
- $\Omega_1 \leftouterjoin \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$.

Sets of mappings

$$\Omega_1 = \begin{array}{|l|l|} \hline ?X & ?Name \\ \hline \_:a & bob \\ alice.org\#me & alice \\ \_:c & bob \\ \hline \end{array} \qquad \Omega_2 = \begin{array}{|l|l|} \hline ?X & ?Friend \\ \hline \_:a & \_:b \\ alice.org\#me & \_:c \\ \hline \end{array}$$

Examples

- $\Omega_1 \bowtie \Omega_2$,
- $\Omega_1 \cup \Omega_2$,
- $\Omega_1 \setminus \Omega_2$,
- $\Omega_1 \mathbin{⟕} \Omega_2$.

Graph pattern semantics

Let $[\![\cdot]\!]_D$ be a function from graph patterns to sets of mappings, where D is a RDF data set over T.
Let t be a triple pattern and P_1, P_2 graph patterns.

- $[\![t]\!]_D = \{\mu \mid dom(\mu) = var(t) \text{ and } \mu(t) \in D\}$, where $var(t)$ is the set variables occurring in t,
- $[\![(P_1 \text{ AND } P_2)]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$,
- $[\![(P_1 \text{ OPT } P_2)]\!]_D = [\![P_1]\!]_D \mathbin{⟕} [\![P_2]\!]_D$,
- $[\![(P_1 \text{ UNION } P_2)]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D$.

RDF dataset D:

```
(B1, name, paul),              (B1, phone, 777-3426),
(B2, name, john),              (B2, email, john@acd.edu),
(B3, name, george),            (B3, webPage,www.george.edu),
(B4, name, ringo),             (B4, email, ringo@acd.edu),
(B4, webPage, www.starr.edu), (B4, phone, 888-4537).
```

Examples

```
P1 = ((?A, email, ?E) OPT (?A, webPage, ?W)).
P2 = (((?A, name, ?N) OPT (?A, email, ?E)) OPT (?A, webPage, ?W)).
P3 = ((?A, name, ?N) OPT ((?A, email, ?E) OPT (?A, webPage, ?W))).
P4 = ((?A, name, ?N) AND ((?A, email, ?E) UNION (?A, webPage, ?W))).
```

FILTER expression semantics

Let μ be a mapping and R a built-in condition.
μ satisfies R, $\mu \models R$, if

- R is bound($?X$) and $?X \in dom(\mu)$,
- R is $?X = c, ?X \in dom(\mu)$ and $\mu(?X) = c$,
- R is $?X = ?Y$, $?X \in dom(\mu), ?Y \in dom(\mu)$ and $\mu(?X) = \mu(?Y)$,
- R is $(\neg R_1)$, R_1 is a built-in condition and it is not the case that $\mu \models R_1$,[1]
- R is $(R_1 \vee R_2)$, R_1 and R_2 are built-in conditions and $\mu \models R_1$ or $\mu \models R_2$,
- R is $(R_1 \wedge R_2)$, R_1 and R_2 are built-in conditions and $\mu \models R_1$ and $\mu \models R_2$.

Given an RDF dataset D and a FILTER expression $(P \text{ FILTER } R)$.

$$[\![(P \text{ FILTER } R)]\!]_D = \{\mu \in [\![P]\!]_D \mid \mu \models R\}.$$

[1] "$\neg$" is also written "!".

RDF dataset D:

```
(B1, name, paul),              (B1, phone, 777-3426),
(B2, name, john),              (B2, email, john@acd.edu),
(B3, name, george),            (B3, webPage,www.george.edu),
(B4, name, ringo),             (B4, email, ringo@acd.edu),
(B4, webPage, www.starr.edu), (B4, phone, 888-4537).
```

Examples

```
P5 = (((?A, name, ?N) OPT (?A, phone, ?P)) FILTER  !bound(?P)).
```