Azzolini Riccardo 2020-05-19

SPARQL

1 Linguaggi di interrogazione per RDF/RDFS

Dal momento che RDF/RDFS hanno una serializzazione in XML, per le interrogazioni si potrebbe pensare di utilizzare, ad esempio, XPath. I linguaggi di interrogazione per XML, però, non sfruttano le definizioni (sintassi e significato) dei termini RDF/RDFS: in pratica, essi "vedono" solo gli elementi XML della serializzazione, senza avere alcuna "comprensione" del modello dei dati.

Per questo motivo, è stato necessario definire dei linguaggi di interrogazione appositi per RDF.

2 SPARQL

SPARQL (pronunciato "sparkle"), *SPARQL Protocol And RDF Query Language*, è un linguaggio di interrogazione per RDF/RDFS, standardizzato dal W3C. Esso ha una sintassi "SQL-like" (simile a SQL), e si basa sul concetto di **matching di pattern di grafi**.

In generale, un **pattern di grafo** è un grafo labellato (etichettato – nel caso di RDF, con etichette sia sui nodi che sugli archi) nel quale le label possono essere non solo risorse, ma anche variabili.

Il più semplice pattern di grafo è un **pattern di tripla**, cioè appunto una tripla nella quale, al posto delle risorse (soggetto, predicato e/o oggetto), possono comparire delle variabili. Un esempio è:

$$x \text{ insegna MIGD}$$

Per eseguire il processo di matching, vengono controllate tutte le triple esistenti, e selezionate tutte quelle che "combaciano" ("match") sulle risorse specificate nel pattern (nell'esempio, "insegna" e "MIGD").

3 Esempi di interrogazioni

- Un esempio di interrogazione SPARQL basata su un pattern di tripla è la seguente, che restituisce tutte le classi (RDFS):

```
SELECT ?c
WHERE {
  ?c rdf:type rdfs:Class .
}
```

  - `?c` è una variabile. In generale, in SPARQL, i nomi delle variabili iniziano con il punto di domanda.
  - Nella clausola `WHERE` è specificato un singolo pattern di tripla:

    `?c rdf:type rdfs:Class .`

    Esso combacia con tutte le triple composte da un soggetto qualunque, dal predicato `rdf:type` (che, come già visto, indica la relazione di appartenenza di un'istanza a una classe, $\in$) e dall'oggetto `rdfs:Class` (la risorsa che descrive il concetto di classe RDFS). Le risorse che sono soggetti di queste triple vengono legate alla variabile `?c`.
  - La clausola `SELECT` specifica quali valori verranno restituiti dall'interrogazione (cioè esegue una *proiezione*, esattamente come la clausola `SELECT` di SQL). Qui vengono restituiti i valori legati alla variabile `?c`, cioè i soggetti delle triple individuate dal pattern.
  - È prevista anche una clausola opzionale `FROM`. Se omessa, come in questo caso, il grafo da interrogare si considera specificato implicitamente (tipicamente, è il grafo del database nel quale si esegue l'interrogazione).
  - Per semplicità, è stato omesso anche il preambolo, nel quale si definiscono gli URI dei namespace usati (qui `rdf` e `rdfs`).

- Un altro esempio di interrogazione simile è quella che restituisce tutte le istanze della classe `uni:course` (che si suppone rappresenti i corsi di un'università):

```
SELECT ?x
WHERE {
  ?x rdf:type uni:course .
}
```

  Se il query processor SPARQL usato per eseguire quest'interrogazione supporta le regole di inferenza di RDFS, verranno restituite anche le istanze delle eventuali sottoclassi di `uni:course`. Diversi query processor hanno diversi livelli di supporto delle regole di inferenza: alcuni non le supportano proprio, altri ne supportano solo alcune, mentre altri ancora le supportano tutte (o quasi).

- La congiunzione di uno o più pattern di triple è chiamata **basic graph pattern**. Ad esempio, per ottenere tutti gli `uni:lecturer` e i loro numeri di telefono:

```
SELECT ?lecturer ?number
WHERE {
  ?lecturer rdf:type uni:lecturer .
  ?lecturer uni:phone ?number .
}
```

  Più pattern di tripla con lo stesso soggetto possono essere anche espressi mediante una sintassi abbreviata, come nella serializzazione Turtle di RDF:

```
SELECT ?lecturer ?number
WHERE {
  ?lecturer rdf:type uni:lecturer ;
            uni:phone ?number .
}
```

  Quando quest'interrogazione viene valutata:

  1. Il primo pattern di tripla,

     ```
     ?lecturer rdf:type uni:lecturer .
     ```

     recupera tutte le istanze della classe `uni:lecturer`, e le lega alla variabile `?lecturer`.

  2. Il secondo pattern di tripla,

     ```
     ?lecturer uni:phone ?number .
     ```

     recupera tutte le triple con il predicato `uni:phone` tra quelle che hanno come soggetto valori legati a `?lecturer`, e lega gli oggetti alla variabile `?number`.

  3. Come indicato nella clausola `SELECT`, vengono restituiti i valori di `?lecturer` e `?number` individuati ai passi precedenti.

  Di fatto, questa query esegue un *join implicito* tra triple.

- All'interno di un pattern di grafo è possibile specificare anche condizioni booleane, mediante la clausola `FILTER`. Ad esempio, essa può essere usata per esprimere un *join esplicito*, come nella seguente interrogazione, che restituisce i nomi dei corsi insegnati dal lecturer che ha l'ID `12345`:

```
SELECT ?name
WHERE {
  ?x rdf:type uni:course ;
     uni:isTaughtBy :12345 .
  ?c uni:name ?name .
```

```
  FILTER (?c = ?x)
}
```

1. Il pattern

   ```
   ?x rdf:type uni:course ;
      uni:isTaughtBy :12345 .
   ```

   lega alla variabile `?x` tutti i corsi (istanze della classe `uni:course`) insegnati (`uni:isTaughtBy`) dal lecturer con ID `12345`.

2. Il pattern

   ```
   ?c uni:name ?name .
   ```

   lega a `?c` tutte le risorse che hanno una proprietà `uni:name`, e lega a `?name` il valore di tale proprietà (cioè l'oggetto della tripla).

3. La clausola

   ```
   FILTER (?c = ?x)
   ```

   seleziona solo i risultati del matching dei pattern che legano alle variabili `?c` e `?x` la stessa risorsa, realizzando un *join esplicito*.

4. Il `SELECT` indica di restituire solo i valori di `?name`.

La stessa query potrebbe invece essere scritta con un join implicito, semplicemente usando la stessa variabile `?x` anche come soggetto dell'ultimo pattern di tripla:

```
SELECT ?name
WHERE {
  ?x rdf:type uni:course ;
     uni:isTaughtBy :12345 ;
     uni:name ?name .
}
```