Linked Open Data e Semantic Web: Fondamenti e Linguaggi di Interrogazione Parte Seconda

Cristiano Longo
longo@dmi.unict.it

Università di Catania, 11/10/2014

Ontologie - Sintassi

Siano N_C, N_P, N_I tre insiemi infiniti, numerabili e a due a due disgiunti di nomi di *classe*, *proprietà* e *individuo*, rispettivamente.

Una *ontologia* è un insieme finito di asserzioni dei seguenti tipi:

(Constraints)	$C \sqsubseteq D$ $P \sqsubseteq Q$ $\mathrm{dom}(P) \sqsubseteq C$ $\mathrm{range}(P) \sqsubseteq C$
(Class Assertions)	$C(a)$
Property Assertions	$a\,P\,b$ (equivalente $P(a, b)$)

dove $C, D \in N_C$, $P, Q \in N_P$ e $a, b \in N_I$.

Ontologie - Semantica

Una *interpretazione* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ è una coppia $\Delta^{\mathcal{I}}$, $\cdot^{\mathcal{I}}$ dove:

- $\Delta^{\mathcal{I}}$ è un insieme non vuoto;
- $\cdot^{\mathcal{I}}$ è una funzione (polimorfa) che associa
  - ad ogni nome di concetto C in N_C un sottoinsieme $C^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$,
  - ad ogni nome di proprietà in P in N_P una relazione $P^{\mathcal{I}}$su $\Delta^{\mathcal{I}}$,
  - ad ogni nome di individuo a in N_I un elemento $a^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$.

La nozione di *soddisfacibilità* è definita come segue ($\mathcal{I} \models \alpha$ si legge $\mathcal{I}$ soddisfa α):

$$\begin{array}{rcl}
\mathcal{I} \models C \sqsubseteq D & \iff & C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\
\mathcal{I} \models P \sqsubseteq Q & \iff & P^{\mathcal{I}} \subseteq Q^{\mathcal{I}} \\
\mathcal{I} \models \mathrm{dom}(P) \sqsubseteq C & \iff & (\forall [x,y] \in P^{\mathcal{I}})(x \in C^{\mathcal{I}}) \\
\mathcal{I} \models \mathrm{range}(P) \sqsubseteq C & \iff & (\forall [x,y] \in P^{\mathcal{I}})(y \in C^{\mathcal{I}}) \\
\mathcal{I} \models C(a) & \iff & a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \models a\, P\, b & \iff & [a^{\mathcal{I}}, b^{\mathcal{I}}] \in P^{\mathcal{I}}
\end{array}$$

per ogni $C, D \in N_C$, $P, Q \in PNames$, $a, b \in N_I$.

Sia $\mathcal{O} = \{\alpha_1, \ldots, \alpha_n\}$ una ontologia.

$$\mathcal{I} \models \mathcal{O} \iff \mathcal{I} \models \alpha_i \text{ per ogni } 1 \leq i \leq n.$$

Siano $\mathcal{O}$ e $\mathcal{O}'$ due ontologie. $\mathcal{O}$ *implica* $\mathcal{O}'$ sse

$$\mathcal{I} \models \mathcal{O} \Longrightarrow \mathcal{I} \models \mathcal{O}'$$

per ogni possibile interpretazione $\mathcal{I}$.

Ontologie - Semantica

Una *interpretazione* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ è una coppia $\Delta^{\mathcal{I}}$, $\cdot^{\mathcal{I}}$ dove:

- $\Delta^{\mathcal{I}}$ è un insieme non vuoto;
- $\cdot^{\mathcal{I}}$ è una funzione (polimorfa) che associa
  - ad ogni nome di concetto C in N_C un sottoinsieme $C^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$,
  - ad ogni nome di proprietà in P in N_P una relazione $P^{\mathcal{I}}$su $\Delta^{\mathcal{I}}$,
  - ad ogni nome di individuo a in N_I un elemento $a^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$.

La nozione di *soddisfacibilità* è definita come segue ($\mathcal{I} \models \alpha$ si legge $\mathcal{I}$ soddisfa α):

$$\begin{array}{rcl}
\mathcal{I} \models C \sqsubseteq D & \iff & C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\
\mathcal{I} \models P \sqsubseteq Q & \iff & P^{\mathcal{I}} \subseteq Q^{\mathcal{I}} \\
\mathcal{I} \models \mathrm{dom}(P) \sqsubseteq C & \iff & (\forall [x, y] \in P^{\mathcal{I}})(x \in C^{\mathcal{I}}) \\
\mathcal{I} \models \mathrm{range}(P) \sqsubseteq C & \iff & (\forall [x, y] \in P^{\mathcal{I}})(y \in C^{\mathcal{I}}) \\
\mathcal{I} \models C(a) & \iff & a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \models a\, P\, b & \iff & [a^{\mathcal{I}}, b^{\mathcal{I}}] \in P^{\mathcal{I}}
\end{array}$$

per ogni $C, D \in N_C$, $P, Q \in PNames$, $a, b \in N_I$.

Sia $\mathcal{O} = \{\alpha_1, \ldots, \alpha_n\}$ una ontologia.

$$\mathcal{I} \models \mathcal{O} \iff \mathcal{I} \models \alpha_i \text{ per ogni } 1 \leq i \leq n.$$

Siano $\mathcal{O}$ e $\mathcal{O}'$ due ontologie. $\mathcal{O}$ *implica* $\mathcal{O}'$ sse

$$\mathcal{I} \models \mathcal{O} \Longrightarrow \mathcal{I} \models \mathcal{O}'$$

per ogni possibile interpretazione $\mathcal{I}$.

Ontologie - Semantica

Una *interpretazione* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ è una coppia $\Delta^{\mathcal{I}}$, $\cdot^{\mathcal{I}}$ dove:

- $\Delta^{\mathcal{I}}$ è un insieme non vuoto;
- $\cdot^{\mathcal{I}}$ è una funzione (polimorfa) che associa
  - ad ogni nome di concetto C in N_C un sottoinsieme $C^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$,
  - ad ogni nome di proprietà in P in N_P una relazione $P^{\mathcal{I}}$su $\Delta^{\mathcal{I}}$,
  - ad ogni nome di individuo a in N_I un elemento $a^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$.

La nozione di *soddisfacibilità* è definita come segue ($\mathcal{I} \models \alpha$ si legge $\mathcal{I}$ soddisfa α):

$$\begin{array}{rcl}
\mathcal{I} \models C \sqsubseteq D & \iff & C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\
\mathcal{I} \models P \sqsubseteq Q & \iff & P^{\mathcal{I}} \subseteq Q^{\mathcal{I}} \\
\mathcal{I} \models \mathrm{dom}(P) \sqsubseteq C & \iff & (\forall [x, y] \in P^{\mathcal{I}})(x \in C^{\mathcal{I}}) \\
\mathcal{I} \models \mathrm{range}(P) \sqsubseteq C & \iff & (\forall [x, y] \in P^{\mathcal{I}})(y \in C^{\mathcal{I}}) \\
\mathcal{I} \models C(a) & \iff & a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \models a\, P\, b & \iff & [a^{\mathcal{I}}, b^{\mathcal{I}}] \in P^{\mathcal{I}}
\end{array}$$

per ogni $C, D \in N_C$, $P, Q \in PNames$, $a, b \in N_I$.

Sia $\mathcal{O} = \{\alpha_1, \ldots, \alpha_n\}$ una ontologia.

$$\mathcal{I} \models \mathcal{O} \iff \mathcal{I} \models \alpha_i \text{ per ogni } 1 \leq i \leq n.$$

Siano $\mathcal{O}$ e $\mathcal{O}'$ due ontologie. $\mathcal{O}$ *implica* $\mathcal{O}'$ sse

$$\mathcal{I} \models \mathcal{O} \Longrightarrow \mathcal{I} \models \mathcal{O}'$$

per ogni possibile interpretazione $\mathcal{I}$.

Ontologie - Semantica

Una *interpretazione* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ è una coppia $\Delta^{\mathcal{I}}$, $\cdot^{\mathcal{I}}$ dove:

- $\Delta^{\mathcal{I}}$ è un insieme non vuoto;
- $\cdot^{\mathcal{I}}$ è una funzione (polimorfa) che associa
  - ad ogni nome di concetto C in N_C un sottoinsieme $C^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$,
  - ad ogni nome di proprietà in P in N_P una relazione $P^{\mathcal{I}}$su $\Delta^{\mathcal{I}}$,
  - ad ogni nome di individuo a in N_I un elemento $a^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$.

La nozione di *soddisfacibilità* è definita come segue ($\mathcal{I} \models \alpha$ si legge $\mathcal{I}$ soddisfa α):

$$\begin{array}{rcl}
\mathcal{I} \models C \sqsubseteq D & \iff & C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\
\mathcal{I} \models P \sqsubseteq Q & \iff & P^{\mathcal{I}} \subseteq Q^{\mathcal{I}} \\
\mathcal{I} \models \mathrm{dom}(P) \sqsubseteq C & \iff & (\forall [x, y] \in P^{\mathcal{I}})(x \in C^{\mathcal{I}}) \\
\mathcal{I} \models \mathrm{range}(P) \sqsubseteq C & \iff & (\forall [x, y] \in P^{\mathcal{I}})(y \in C^{\mathcal{I}}) \\
\mathcal{I} \models C(a) & \iff & a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \models a\, P\, b & \iff & [a^{\mathcal{I}}, b^{\mathcal{I}}] \in P^{\mathcal{I}}
\end{array}$$

per ogni $C, D \in N_C$, $P, Q \in PNames$, $a, b \in N_I$.

Sia $\mathcal{O} = \{\alpha_1, \ldots, \alpha_n\}$ una ontologia.

$$\mathcal{I} \models \mathcal{O} \iff \mathcal{I} \models \alpha_i \text{ per ogni } 1 \leq i \leq n.$$

Siano $\mathcal{O}$ e $\mathcal{O}'$ due ontologie. $\mathcal{O}$ *implica* $\mathcal{O}'$ sse

$$\mathcal{I} \models \mathcal{O} \Longrightarrow \mathcal{I} \models \mathcal{O}'$$

per ogni possibile interpretazione $\mathcal{I}$.

Ontologie nel Web Semantico

Le ontologie definite usando tecnologie del Web Semantico hanno particolari caratteristiche:

- tutti i *nomi* sono degli *Internationalized Resource Identifier (IRI)*), ossia

$$N_C \cup N_P \cup N_I \subseteq IRI;$$

- possono contenere dei *letterali*, che vengono usati per rappresentare tipi di dato *concreti* (ad esempio stringhe di testo, numeri, date, . . .).

Ontologie nel Web Semantico

Le ontologie definite usando tecnologie del Web Semantico hanno particolari caratteristiche:

- tutti i *nomi* sono degli *Internationalized Resource Identifier (IRI))*, ossia

$$N_C \cup N_P \cup N_I \subseteq IRI;$$

- possono contenere dei *letterali*, che vengono usati per rappresentare tipi di dato *concreti* (ad esempio stringhe di testo, numeri, date, . . .).

International Resource Identifier

La specifica *International Resource Identifier* (in breve *IRI*, vedi RFC3987) estende quella per gli *Uniform Resource Identifier* (in breve *URI*, vedi RFC3986) con un più ampio repertorio di caratteri, aggiungendo funzionalità per l'internazionalizzazione.

Nel seguito indicheremo con *IRI* l'insieme di tutti i possibili IRI (i.e. l'insieme delle stringhe che rispettano la specifica IRI).

IRI nel Web Semantico

Nell'ambito del Web Semantico, tutti gli oggetti reali o concreti sono identificati attraverso IRI. As esempio

`http://dbpedia.org/resource/Leonardo_da_Vinci`

è la IRI usata nell'ontologia `dbpedia.org` per indicare Leonardo da Vinci, e ancora

`http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619`

indica la *Mona Lisa* nell'ontologia del progetto *Europeana*.

IRI nel Web Semantico - *Sinonimie*

Si noti che una IRI è associata ad un unico oggetto, ma ad ogni oggetto possono essere associati diversi IRI.

Ad esempio le seguenti IRI sono entrambe associate alla Monnalisa:

`http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619`

`http://dbpedia.org/page/Mona_Lisa` .

IRI nel Web Semantico - Namespaces

Nelle ontologie del Web Semantico le IRI possono essere abbreviate con il meccanismo dei *namespace* (prefissi), mutuato da XML.

Ad ogni ontologia spesso è assegnato un *base prefix*. Viene usato come prefisso per ottenere le IRI complete degli oggetti dell'ontologia nel caso in cui la IRI specificata sia *incompleta*. Ad esempio, se il base prefix dell'ontologia $\mathcal{O}$ è `http://example.org/`,

$$Alice \quad \Longrightarrow \texttt{http://example.org/Alice}.$$

È possibile specificare degli ulteriori *prefix* come coppie

$$< prefixname > \quad \longrightarrow \quad < prefixuri >$$

e *abbreviare* delle IRI nell'ontologia con la sintassi
$< prefixname >:< IRIspecificpart >$.

Se ad esempio nell'ontologia è definito il prefisso

$$ex2 \quad \longrightarrow \quad \texttt{http://example2.org/}$$

la IRI `ex2:Alice` verrà espansa in `http://example2.org/Alice` .

IRI nel Web Semantico - Namespaces

Nelle ontologie del Web Semantico le IRI possono essere abbreviate con il meccanismo dei *namespace* (prefissi), mutuato da XML.

Ad ogni ontologia spesso è assegnato un *base prefix*. Viene usato come prefisso per ottenere le IRI complete degli oggetti dell'ontologia nel caso in cui la IRI specificata sia *incompleta*. Ad esempio, se il base prefix dell'ontologia $\mathcal{O}$ è `http://example.org/`,

$$Alice \quad \Longrightarrow \texttt{http://example.org/Alice}.$$

È possibile specificare degli ulteriori *prefix* come coppie

$$< prefixname > \quad \longrightarrow \quad < prefixuri >$$

e *abbreviare* delle IRI nell'ontologia con la sintassi
$< prefixname >:< IRIspecificpart >$.

Se ad esempio nell'ontologia è definito il prefisso

$$ex2 \quad \longrightarrow \quad \texttt{http://example2.org/}$$

la IRI `ex2:Alice` verrà espansa in `http://example2.org/Alice` .

IRI nel Web Semantico - Namespaces

Nelle ontologie del Web Semantico le IRI possono essere abbreviate con il meccanismo dei *namespace* (prefissi), mutuato da XML.

Ad ogni ontologia spesso è assegnato un *base prefix*. Viene usato come prefisso per ottenere le IRI complete degli oggetti dell'ontologia nel caso in cui la IRI specificata sia *incompleta*. Ad esempio, se il base prefix dell'ontologia $\mathcal{O}$ è `http://example.org/`,

$$Alice \quad \Longrightarrow \texttt{http://example.org/Alice}.$$

È possibile specificare degli ulteriori *prefix* come coppie

$$< prefixname > \quad \longrightarrow \quad < prefixuri >$$

e *abbreviare* delle IRI nell'ontologia con la sintassi
$< prefixname >:< IRIspecificpart >$.

Se ad esempio nell'ontologia è definito il prefisso

$$ex2 \quad \longrightarrow \quad \texttt{http://example2.org/}$$

la IRI `ex2:Alice` verrà espansa in `http://example2.org/Alice` .

Letterali

I letterali vengono usati per rappresentare tipi di dato *concreti*, come ad esempio stringhe di testo, numeri, date, ...

Grazie ai letterali è possibile ad esempio esprimere affermazioni dei seguenti tipi:

- Il cognome di Mario è *Rossi*;
- Cristiano è nato il giorno *22 Marzo 1979*;
- L'Empire State Building è alto *380 metri*.

Datatype

Per introdurre i Letterali è necessario fornire prima la definizione di *datatype* (vedi `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-Datatypes` e `http://www.w3.org/TR/xmlschema11-2/`).

Un *datatype* è caratterizzato da tre componenti:

- un *lexical space*, ossia un insieme di stringhe (finite) di caratteri nella codifica *UNICODE*;
- un *value space*, che è un insieme non meglio specificato e numerabile di *valori* (interi, date, Booleani, . . .);
- un *lexical-value mapping* che associa ad ogni stringa nel lexical space un elemento nel value space.

I datatype vengono di solito indicati con delle IRI.

Datatype - Esempio 1 : `xsd:integer`

Il datatype `xsd:integer` (dove `xsd` è l'abbreviazione per il namespace `http://www.w3.org/2001/XMLSchema#`) è definito come segue:

- il *lexical space* di `xsd:integer` è costituito da tutte le sequenze finite di cifre da 0 a 9, possibilmente precedute dal carattere "-" o "+";
- il *value space* è l'insieme dei numeri interi;
- il valore di una stringa nel lexical space di `xsd:integer` si ottiene considerando le cifre presenti nella stringa come cifre del corrispondente numero in base 10, e moltiplicando il numero così ottenuto per -1 nel caso in cui la stringa inizi con il carattere "-".

Datatype - Esempio 2 : `xsd:string`

Il datatype `xsd:string` è definito come segue:

- il *lexical space* di `xsd:string` comprende tutte le sequenze di caratteri (UNICODE) di zero o più caratteri;
- il *value space* di `xsd:string` coincide col suo lexical space;
- il *lexical-value mapping* associa ogni stringa nel lexical space con se stessa (indetità).

Altri esempi di datatype

Riportiamo alcuni datatype (mutuati da XML Schema) di uso comune.

`xsd:boolean`	true, false
`xsd:decimal`	Arbitrary-precision decimal numbers
`xsd:integer`	Arbitrary-size integer numbers
`xsd:double`	64-bit floating point numbers incl. $\pm$Inf, $\pm$0, NaN
`xsd:float`	32-bit floating point numbers incl. $\pm$Inf, $\pm$0, NaN
`xsd:date`	Dates (yyyy-mm-dd) with or without timezone
`xsd:time`	Times (hh:mm:ss.sss...) with or without timezone
`xsd:dateTime`	Date and time with or without timezone
`xsd:dateTimeStamp`	Date and time with required timezone
`xsd:duration`	Duration of time
`xsd:byte`	-128...+127 (8 bit)
`xsd:short`	-32768...+32767 (16 bit)
`xsd:int`	-2147483648...+2147483647 (32 bit)
`xsd:long`	-9223372036854775808...+9223372036854775807 (64 bit)
`xsd:unsignedByte`	0...255 (8 bit)
`xsd:unsignedShort`	0...65535 (16 bit)
`xsd:unsignedInt`	0...4294967295 (32 bit)
`xsd:unsignedLong`	0...18446744073709551615 (64 bit)
`xsd:positiveInteger`	Integer numbers > 0
`xsd:nonNegativeInteger`	Integer numbers ≥ 0
`xsd:negativeInteger`	Integer numbers < 0
`xsd:nonPositiveInteger`	Integer numbers ≤ 0
`xsd:hexBinary`	Hex-encoded binary data

Letterali - Definizione

Formalmente, i letterali sono definiti come segue (vedi `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-Graph-Literal`).

Un *letterale* è costituito da un *datatype* t e da una stringa di caratteri nel lexical space di t (la cosiddetta *lexical form* del letterale).

Se un letterale è di tipo `xsd:string`, ad esso può essere associato un *language tag* ad indicarne la *lingua*. Per i valori che questo attributo può assumere fare riferimento allo *IANA Language Subtag Registry*.

Letterali - Esempi

Seguono alcuni esempi di letterali:

lexical form	data type	language tag
"380"	`xsd:integer`	-
"March-22-1979"	`xsd:date`	-
"Rossi"	`xsd:string`	-
"Parigi"	`xsd:string`	it
"Paris"	`xsd:string`	en

Nel caso in cui si ometta l'indicazione del tipo di dato, il letterale si assume essere di tipo `xsd:string`.

Letterali - Notazione (1/3)

Per indicare i letterali spesso si usano le seguenti notazioni

"lexform"^^type

$< lexform, type >$

ove *lexform* e *type* sono la lexical form e il data type del letterale, rispettivamente.

Alcuni esempi:

"380"^^xsd:integer	$<$ "380", xsd : integer $>$
"March-22-1979"^^xsd:date	$<$ "*March* − 22 − 1979", xsd : date $>$
"Rossi"^^xsd:string	$<$ "*Rossi*", xsd : string $>$

Letterali - Notazione (3/3)

Dato un letterale l, indicheremo con

- *datatype*(l) il tipo di dato di l, e con
- *lexform*(l) la lexical form di l.

Seguono alcuni esempi:

datatype("380" ^^`xsd:integer`) = `xsd:integer`
datatype("March-22-1979" ^^`xsd:date`) = `xsd:date`
datatype("Rossi" ^^`xsd:string`) = `xsd:string`

lexform("380" ^^`xsd:integer`) = "380"
lexform("March-22-1979" ^^`xsd:date`) = "March-22-1979"
lexform("Rossi" ^^`xsd:string`) = "Rossi"

Confronto tra Letterali

Due letterali sono uguali se e solo se sono uguali le loro lexical form, se hanno lo stesso tipo e se sono uguali i loro language tag, ove presenti. Di conseguenza due letterali possono essere diversi anche avendo lo stesso *valore*. Ad esempio i due seguenti letterali hanno entrambi come valore l'intero 1 ma sono diversi:

"1"^^`xsd:integer`
"01"^^`xsd:integer`.

Property Assertions con Letterali

I letterali possono essere usati per specificare *proprietà* di un individuo. In altre parole, essi possono comparire come *oggetto* di una property assertion. Alcuni esempi di role assertion che coinvolgono letterali sono

Mario surname "Rossi"^^`xsd:string`
Cristiano hasbirth "March-22-1979"^^`xsd:date`

con *Mario*, *Cristiano* $\in N_I$, *surname*, *hasbirth* $\in N_P$ e "Rossi"^^`xsd:string`, "March-22-1979"^^`xsd:date` letterali.

Ontologie nel Web Semantico - Sintassi

Siano N_C, N_P, N_I, N_D, $\mathcal{L}$ insiemi infiniti, numerabili e a due a due disgiunti di nomi di *classe*, *proprietà*, *individuo* e di *datatype* e di *letterali*, rispettivamente, tali che

$$N_C \cup N_P \cup N_I \cup N_D \subseteq IRI,$$

e $\mathcal{L}$ siano i letterali (nel senso visto prima) i cui datatype sono in N_D.
Una *ontologia* è un insieme finito di asserzioni dei seguenti tipi:

(Constraints)	$C \sqsubseteq D$
	$P \sqsubseteq Q$
	$\mathrm{dom}(P) \sqsubseteq C$
	$\mathrm{range}(P) \sqsubseteq C$
(Class Assertions)	$C(a)$
Property Assertions	$a\ P\ b$
	$\mathbf{a\ P\ l}$

dove $C, D \in N_C$, $P, Q \in N_P$, $a, b \in N_I$, e $l \in N_L$.

Ontologie nel Web Semantico - Semantica

Una *interpretazione* è una tripla $\mathcal{I} = (\Delta^{\mathcal{I}}, \Delta_D^{\mathcal{I}}, \cdot^{\mathcal{I}})$ dove:

- $\Delta^{\mathcal{I}}, \Delta_D^{\mathcal{I}}$ sono due insiemi disgiunti e non vuoti;
- $\cdot^{\mathcal{I}}$ è una funzione (polimorfa) che associa
  - ad ogni nome di concetto C in N_C un sottoinsieme $C^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$,
  - ad ogni datatype t un sottoinsieme di $t^{\mathcal{I}}$ di Δ_D,
  - ad ogni nome di proprietà in P in N_P una relazione $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times (\Delta^{\mathcal{I}} \cup \Delta_D^{\mathcal{I}})$,
  - ad ogni nome di individuo a in N_I un elemento $a^{\mathcal{I}}$ di $\Delta^{\mathcal{I}}$,
  - ad ogni letterale l in $\mathcal{L}$ un elemento in $t^{\mathcal{I}}$, ove $t = \mathit{datatype}(l)$.

Le nozione di soddisfacibilità di una ontologia e di implicazione di ontologie seguono da questa nuova definizione.

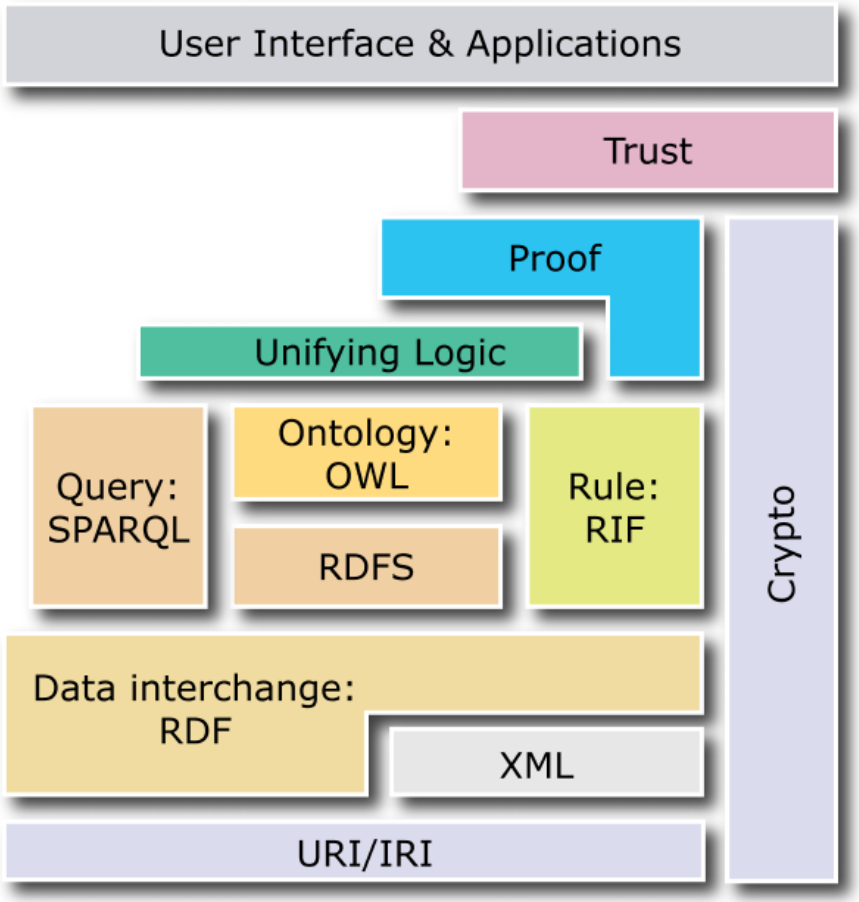
Ontologie nel Web Semantico - Alcune notazioni

Data una qualsiasi ontologia (nel senso appena visto) $\mathcal{O}$, indicheremo con:

- $N_C(\mathcal{O})$ l'insieme dei nomi di concetto che compaiono in $\mathcal{O}$;
- $N_P(\mathcal{O})$ l'insieme dei nomi di proprietà che compaiono in $\mathcal{O}$;
- $N_I(\mathcal{O})$ l'insieme dei nomi di individuo che compaiono in $\mathcal{O}$;
- $\mathcal{L}(\mathcal{O})$ l'insieme dei letterali che compaiono in $\mathcal{O}$.

Semantic Web Stack

L'insieme delle tecnologie usate nel Web Semantico costituiscono il cosiddetto *Semantic Web Stack*.



Il Web Semantico è costituito dall'insieme dei dataset codificati ed esposti attraverso queste tecnologie.

Il protocollo SPARQL

Le basi di conoscenza presenti sul Web Semantico usualmente mettono a disposizione uno *SPARQL endpoint* che permette di interrogarle e, ove permesso, di modificarle.

Knowledge Base	**Endpoint IRI**
Europeana	`http://europeana.ontotext.com/sparql`
CNR	`http://data.cnr.it/sparql/`
Camera dei Deputati	`http://dati.camera.it/sparql`
DBPedia	`http://dbpedia.org/sparql`

Table: Alcuni endpoint sparql

Il *protocollo SPARQL* (vedi `http://www.w3.org/TR/sparql11-protocol/`) è basato sul protocollo HTTP le richieste SPARQL vengono inviate agli endpoint come richieste GET o POST e l'endpoint risponde con un *esito*.

Le richieste si suddividono in richieste di *query* o *update*.
In caso di richieste di tipo query effettuate con successo, la risposta alla chiamata GET o POST conterrà anche tutte le *soluzioni* dell'interrogazione in uno dei formati XML, JSON o CSV (il formato di risposta va specificato nella richesta).

Il protocollo SPARQL

Le basi di conoscenza presenti sul Web Semantico usualmente mettono a disposizione uno *SPARQL endpoint* che permette di interrogarle e, ove permesso, di modificarle.

Knowledge Base	**Endpoint IRI**
Europeana	`http://europeana.ontotext.com/sparql`
CNR	`http://data.cnr.it/sparql/`
Camera dei Deputati	`http://dati.camera.it/sparql`
DBPedia	`http://dbpedia.org/sparql`

Table: Alcuni endpoint sparql

Il *protocollo SPARQL* (vedi `http://www.w3.org/TR/sparql11-protocol/`) è basato sul protocollo HTTP le richieste SPARQL vengono inviate agli endpoint come richieste GET o POST e l'endpoint risponde con un *esito*.

Le richieste si suddividono in richieste di *query* o *update*.
In caso di richieste di tipo query effettuate con successo, la risposta alla chiamata GET o POST conterrà anche tutte le *soluzioni* dell'interrogazione in uno dei formati XML, JSON o CSV (il formato di risposta va specificato nella richesta).

Il protocollo SPARQL

Le basi di conoscenza presenti sul Web Semantico usualmente mettono a disposizione uno *SPARQL endpoint* che permette di interrogarle e, ove permesso, di modificarle.

Knowledge Base	**Endpoint IRI**
Europeana	`http://europeana.ontotext.com/sparql`
CNR	`http://data.cnr.it/sparql/`
Camera dei Deputati	`http://dati.camera.it/sparql`
DBPedia	`http://dbpedia.org/sparql`

Table: Alcuni endpoint sparql

Il *protocollo SPARQL* (vedi `http://www.w3.org/TR/sparql11-protocol/`) è basato sul protocollo HTTP le richieste SPARQL vengono inviate agli endpoint come richieste GET o POST e l'endpoint risponde con un *esito*.

Le richieste si suddividono in richieste di *query* o *update*.
In caso di richieste di tipo query effettuate con successo, la risposta alla chiamata GET o POST conterrà anche tutte le *soluzioni* dell'interrogazione in uno dei formati XML, JSON o CSV (il formato di risposta va specificato nella richesta).

SPARQL Query Language

Le richieste di tipo *query* vanno specificate nel linguaggio denominato *SPARQL Query*. La specifica di questo linguaggio è disponibile all'indirizzo

`http://www.w3.org/TR/sparql11-query/` .

Una *query SPARQL* ha la seguente sintassi

```
BASE <iriBase>
PREFIX p1 : <iriP1>
...
PREFIX pn : <iriPn>

SELECT ?x1 ... ?xm WHERE { GraphPattern }
```

dove:

- la sezione *BASE* $<$ *iriBase* $>$ è opzionale;
- $p1, \ldots, pn$ sono prefissi di namespace ($n \geq 0$, se $n = 0$ non è presente alcun prefisso);
- $<$ *iriP*1 $>, \ldots, <$ *iriPn* $>$ sono IRI;
- $?x1, \ldots, ?xm$ sono *variabili* ($m > 0$);
- *GraphPattern* è un *graph pattern* di uno dei tipi che vedremo in seguito.

SPARQL Query Language - Triple Pattern

Il tipo di graph pattern più basilare è quello dei *triple pattern*. Un triple pattern è una espressione di uno dei seguenti tipi:

$$s\ p\ o\,, \quad s\ \mathtt{a}\ o$$

ove s e p possono essere IRI o variabili, a è una parola riservata, e o può essere una IRI, una variabile o un letterale.

Le formule atomiche nelle query congiuntive possono essere facilmente tradotte in triple patterns:

$$C(?a) \quad \Longrightarrow \quad ?a\ \mathtt{a}\ C \quad | \quad ?a\ \mathtt{rdf:type}\ C \text{(le due formulazioni sono equivalenti)}$$

$$?a\ P\ ?b \quad \Longrightarrow \quad ?a\ P\ ?b.$$

Si noti che la proprietà `rdf:type` viene utilizzata per esprimere l'appartenenza nel linguaggio RDF.

A differenza delle query congiuntive, in SPARQL le variabili possono comparire anche al posto del nome della classe o della proprietà. Ad esempio i seguenti due sono triple pattern validi:

$?C(a)$ "Trova tutte le classi cui appartiene a"

$?a\ ?p\ ?b$ "Trova tutte le triple nell'ontologia".

SPARQL Query Language - Triple Pattern

Il tipo di graph pattern più basilare è quello dei *triple pattern*. Un triple pattern è una espressione di uno dei seguenti tipi:

$$s\ p\ o\,, \quad s\ \mathtt{a}\ o$$

ove s e p possono essere IRI o variabili, a è una parola riservata, e o può essere una IRI, una variabile o un letterale.

Le formule atomiche nelle query congiuntive possono essere facilmente tradotte in triple patterns:

$C(?a) \implies ?a\ \mathtt{a}\ C \quad | \quad ?a\ \mathtt{rdf:type}\ C$ (le due formulazioni sono equivalenti)

$?a\ P\ ?b \implies ?a\ P\ ?b.$

Si noti che la proprietà `rdf:type` viene utilizzata per esprimere l'appartenenza nel linguaggio RDF.

A differenza delle query congiuntive, in SPARQL le variabili possono comparire anche al posto del nome della classe o della proprietà. Ad esempio i seguenti due sono triple pattern validi:

$?C(a)$ "Trova tutte le classi cui appartiene a"

$?a\ ?p\ ?b$ "Trova tutte le triple nell'ontologia".

SPARQL Query Language - Triple Pattern

Il tipo di graph pattern più basilare è quello dei *triple pattern*. Un triple pattern è una espressione di uno dei seguenti tipi:

$$s\ p\ o\,, \quad s\ \mathtt{a}\ o$$

ove s e p possono essere IRI o variabili, a è una parola riservata, e o può essere una IRI, una variabile o un letterale.

Le formule atomiche nelle query congiuntive possono essere facilmente tradotte in triple patterns:

$C(?a) \implies ?a\ \mathtt{a}\ C \quad | \quad ?a\ \mathtt{rdf:type}\ C$ (le due formulazioni sono equivalenti)

$?a\ P\ ?b \implies ?a\ P\ ?b.$

Si noti che la proprietà `rdf:type` viene utilizzata per esprimere l'appartenenza nel linguaggio RDF.

A differenza delle query congiuntive, in SPARQL le variabili possono comparire anche al posto del nome della classe o della proprietà. Ad esempio i seguenti due sono triple pattern validi:

$?C(a)$ "Trova tutte le classi cui appartiene a"

$?a\ ?p\ ?b$ "Trova tutte le triple nell'ontologia".

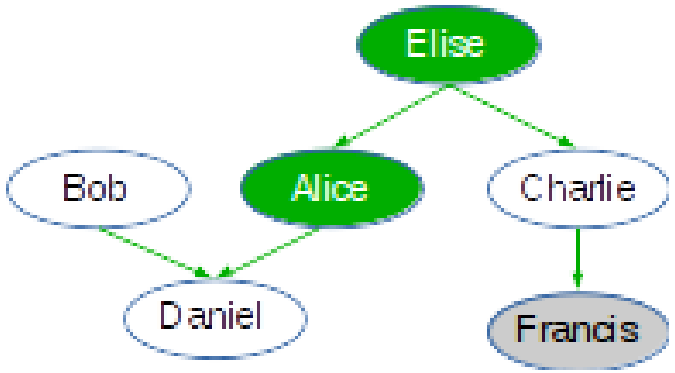
SPARQL Query Language - Soluzioni

Analogamente a quanto accade nell'ambito del conjunctive query answering, le *soluzioni* per un triple pattern T rispetto a una ontologia $\mathcal{O}$ sono tutte le sostituzioni σ che associano ad ogni variabile presente in T una IRI o un letterale in modo tale che σT sia presente in $\mathcal{O}$.

Il risultato di una query Q, avente come pattern un triple pattern T e come variabili nella select $x_1, \ldots, x_n$, rispetto all'ontologia $\mathcal{O}$ è l'insieme delle sostituzioni σ, ristrette alle variabili $x_1, \ldots, x_n$, tali che σT appartiene ad $\mathcal{O}$.

SPARQL Query Language - Triple Pattern - Esempio

Consideriamo ad esempio la seguente ontologia (base prefix `http://ex.org/`)



$$\begin{aligned}\mathcal{O} = \quad & \{Female(Elise), Female(Alice), Male(Bob),\\ & Male(Charlie), Male(Daniel),\\ & Alice\ childOf\ Elise, Charlie\ childOf\ Elise,\\ & Daniel\ childOf\ Alice, Daniel\ childOf\ Bob,\\ & Francis\ childOf\ Charlie\}\end{aligned}$$

Consideriamo inoltre la query Q definita come segue.

```
BASE <http://ex.org/>

SELECT ?x ?y WHERE { ?x childOf ?y . }
```

Le soluzioni di Q rispetto a $\mathcal{O}$ sono

?x	?y
<http://ex.org/Eliza>	<http://ex.org/Giorgia>
<http://ex.org/Eliza>	<http://ex.org/Francis>

SPARQL Query Language - Basic Graph Pattern

Un *basic graph pattern* è un insieme di triple pattern, separati dal carattere ".". Una sostituzione è una soluzione per un basic graph pattern se e solo se è una soluzione per tutti i triple pattern che compaiono in esso.

Un esempio di query con il basic graph pattern è il seguente:

> *"Trova tutte le persone con almeno un figlo maschio"*

```
BASE <http://ex.org/>

SELECT ?x  WHERE {
  ?y childOf ?x .
  ?y a Male
}
```

Da una query congiuntiva si può ottenere il basic graph pattern corrispondente come segue:

- applicare la trasformazione vista in precedenza ai triple pattern che lo costituiscono;
- sostituire i caratteri "." con l'operatore logico di congiunzione $\wedge$.

Ad esempio

$$C(?x) \wedge ?x\ P\ b \quad \Longrightarrow \quad ?x\ \mathrm{a}\ C\ .\ ?x\ P\ b\ .$$

SPARQL Query Language - Basic Graph Pattern

Un *basic graph pattern* è un insieme di triple pattern, separati dal carattere ".". Una sostituzione è una soluzione per un basic graph pattern se e solo se è una soluzione per tutti i triple pattern che compaiono in esso.

Un esempio di query con il basic graph pattern è il seguente:

"Trova tutte le persone con almeno un figlo maschio"

```
BASE <http://ex.org/>

SELECT ?x  WHERE {
  ?y childOf ?x .
  ?y a Male
}
```

Da una query congiuntiva si può ottenere il basic graph pattern corrispondente come segue:

- applicare la trasformazione vista in precedenza ai triple pattern che lo costituiscono;
- sostituire i caratteri "." con l'operatore logico di congiunzione $\wedge$.

Ad esempio

$$C(?x) \wedge ?x\ P\ b \quad \Longrightarrow \quad ?x\ \mathsf{a}\ C\ .\ ?x\ P\ b\ .$$

SPARQL Query Language - Filtri sui letterali

È possibile specificare dei filtri sui letterali che compaiono nei pattern. Essi sono dei predicati, che dipendono dal tipo di dato dei letterali.[1] Permettono di selezionare tutte le soluzioni nelle quali ad una determinata variabile viene associato un letterale che soddisfa un predicato.

La seguente query ad esempio permette di ottenere tutti gli individui con un figlio il cui nome inizia con la lettera 'R'.

```
BASE <http://ex.org/>

SELECT ?x  WHERE {
  ?y childOf ?x .
  ?y fullName ?name .
  FILTER regex(?name, "^R.*") .
}
```

[1] http://www.w3.org/TR/sparql11-query/#OperatorMapping

SPARQL Query Language - Optional Pattern Matching

Dati due graph pattern P_1 e P_2, il seguente è un *optional graph pattern*:

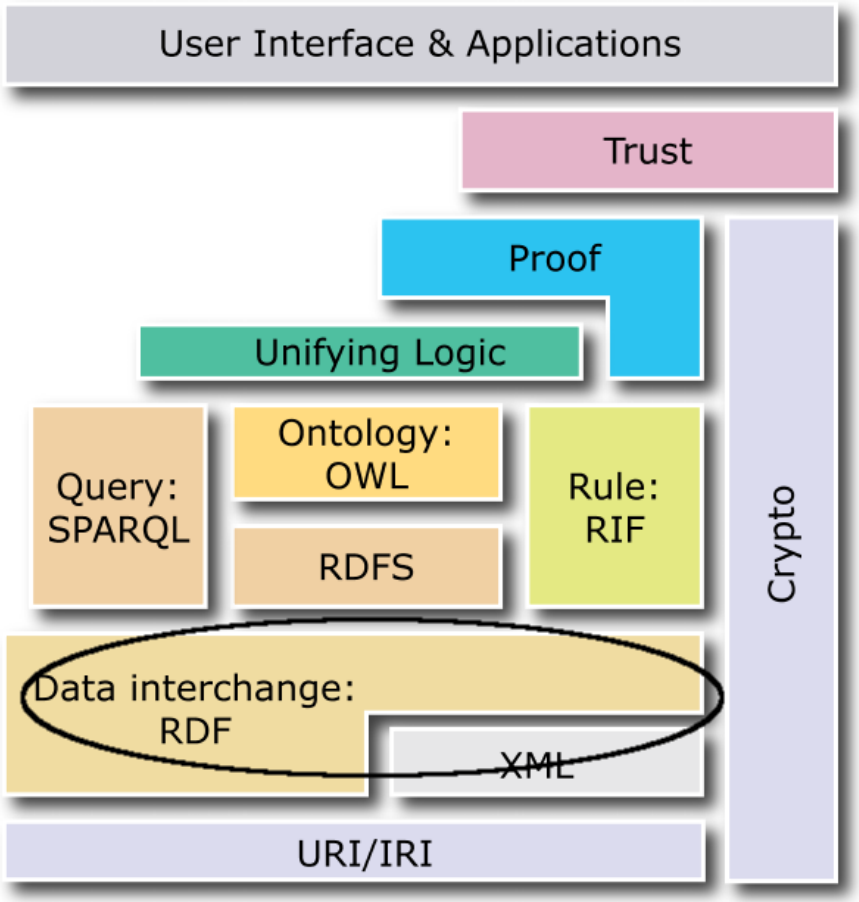
$$P_1 \texttt{OPTIONAL} P_2.$$

Le sostituzioni che sono soluzioni per questo pattern sono quelle che

- sono soluzioni per P_1 e P_2, oppure
- sono soluzioni per P_1

Resource Description Framework (RDF)

Il *linguaggio di rappresentazione* più basilare nella pila delle tecnologie del Web Semantico è il cosiddetto *Resource Description Framework* (nel seguito *RDF*).



Grafi RDF

Le ontologie RDF sono dette *grafi RDF* in quanto possono contenere solo property assertions.

Un *grafo RDF* è un insieme finito di property assertions dei seguenti tipi:

$$\begin{array}{lll} a & P & b \\ a & P & l \end{array}$$

con

$$\begin{array}{l} P \in IRI \\ a, b \in IRI \cup \mathcal{B} \\ l \in \mathcal{L} \end{array}$$

per qualche insieme $\mathcal{B}$ disgiunto da IRI.

Dato un grafo RDf G, gli elementi di $\mathcal{B}$ che compaiono in G sono definiti *blank node* di G. Indicheremo con $\mathcal{B}(G)$ l'insieme dei blank node del grafo G.

Grafi RDF

Le ontologie RDF sono dette *grafi RDF* in quanto possono contenere solo property assertions.

Un *grafo RDF* è un insieme finito di property assertions dei seguenti tipi:

$$\begin{array}{lll} a & P & b \\ a & P & l \end{array}$$

con

$$\begin{array}{l} P \in IRI \\ a, b \in IRI \cup \mathcal{B} \\ l \in \mathcal{L} \end{array}$$

per qualche insieme $\mathcal{B}$ disgiunto da IRI.

Dato un grafo RDf G, gli elementi di $\mathcal{B}$ che compaiono in G sono definiti *blank node* di G. Indicheremo con $\mathcal{B}(G)$ l'insieme dei blank node del grafo G.

Grafi RDF

Le ontologie RDF sono dette *grafi RDF* in quanto possono contenere solo property assertions.

Un *grafo RDF* è un insieme finito di property assertions dei seguenti tipi:

$$\begin{array}{lll} a & P & b \\ a & P & l \end{array}$$

con

$$\begin{array}{l} P \in IRI \\ a, b \in IRI \cup \mathcal{B} \\ l \in \mathcal{L} \end{array}$$

per qualche insieme $\mathcal{B}$ disgiunto da IRI.

Dato un grafo RDf G, gli elementi di $\mathcal{B}$ che compaiono in G sono definiti *blank node* di G. Indicheremo con $\mathcal{B}(G)$ l'insieme dei blank node del grafo G.

Grafi RDF - Esempio

Un esempio di grafo RDF è il seguente:

$$
\begin{aligned}
G \quad =_{\text{Def}} \quad & \{\texttt{ex:Alice ex:childOf ex:Eliza},\\
& \texttt{ex:Bob ex:childOf ex:Eliza},\\
& \texttt{ex:Eliza ex:fullName} \text{ “Eliza Smith”},\\
& \texttt{ex:Eliza ex:age} \text{ “63”}\texttt{\^{}\^{}xsd:nonNegativeInteger}\},
\end{aligned}
$$

RDF - Semantica

Esistono alcuni nomi di proprietà, di classi e di individui che ricoprono dei ruoli *particolari* nel linguaggio RDF:

$$\texttt{rdf:Property}, \texttt{rdf:List} \qquad \in N_C;$$
$$\texttt{rdf:type}, \texttt{rdf:subject}, \texttt{rdf:predicate}, \texttt{rdf:object},$$

$$\texttt{rdf:first}, \texttt{rdf:rest}, \texttt{rdf:value}, \texttt{rdf:\_1}, \texttt{rdf:\_2}, \ldots \qquad \in N_P;$$

$$\texttt{rdf:nil} \qquad \in N_I;$$

$$\texttt{rdf:XMLLiteral} \qquad \in N_D.$$

dove il prefisso `rdf:` è una abbreviazione per il namespace

`http://www.w3.org/1999/02/22-rdf-syntax-ns#`

Una *interpretazione RDF* pone dei vincoli semantici particolari sull'interpretazione di questi termini (ne vedremo solo alcuni).

Class assertions in RDF

Asserzioni del tipo a `rdf:type` C vengono usate in RDF per affermare class assertions $C(a)$.

$$\mathcal{I} \models a \texttt{ rdf:type } C \Leftrightarrow C^{\mathcal{I}}(a^{\mathcal{I}}) \quad (*)$$

In altre parole, da ogni asserzione del tipo a `rdf:type` C tramite reasoning viene inferita una asserzione $C(a)$.

Ad esempio:

```
{ex:Eliza rdf:type ex:Person,
 ex:Alice rdf:type ex:Person,
 ex:Bob rdf:type ex:Person,
 ex:childOf rdf:type rdf:Property,
 ex:fullName rdf:type rdf:Property,
 ex:age rdf:type rdf:Property,
 ex:Alice ex:childOf ex:Eliza,
 ex:Bob ex:childOf ex:Eliza,
 ex:Eliza ex:fullName "Eliza Smith",
 ex:Eliza ex:age "63"^^xsd:nonNegativeInteger },
```

$\Longrightarrow$

```
{ex:Person(ex:Eliza),
 ex:Person(ex:Alice),
 ex:Person(ex:Bob),
 ex:Property(ex:childOf),
 ex:Property(ex:fullName),
 ex:Property(ex:age), }
 ... }
```

Class assertions in RDF

Asserzioni del tipo a `rdf:type` C vengono usate in RDF per affermare class assertions $C(a)$.

$$\mathcal{I} \models a \texttt{ rdf:type } C \Leftrightarrow C^{\mathcal{I}}(a^{\mathcal{I}}) \quad (*)$$

In altre parole, da ogni asserzione del tipo a `rdf:type` C tramite reasoning viene inferita una asserzione $C(a)$.

Ad esempio:

```
{ex:Eliza rdf:type ex:Person,
 ex:Alice rdf:type ex:Person,
 ex:Bob rdf:type ex:Person,
 ex:childOf rdf:type rdf:Property,
 ex:fullName rdf:type rdf:Property,
 ex:age rdf:type rdf:Property,
 ex:Alice ex:childOf ex:Eliza,
 ex:Bob ex:childOf ex:Eliza,
 ex:Eliza ex:fullName "Eliza Smith",
 ex:Eliza ex:age "63"^^xsd:nonNegativeInteger },
```

$\Longrightarrow$

```
{ex:Person(ex:Eliza),
 ex:Person(ex:Alice),
 ex:Person(ex:Bob),
 ex:Property(ex:childOf),
 ex:Property(ex:fullName),
 ex:Property(ex:age), }
 ... }
```

La classe `rdf:Property`

Ogni proprietà P presente in un grafo RDF deve essere una istanza della classe `rdf:Property`

$$\mathcal{I} \models a\, P\, b \iff [a^{\mathcal{I}}, b^{\mathcal{I}}] \in P^{\mathcal{I}} \wedge P^{\mathcal{I}} \in (\texttt{rdf:Property})^{\mathcal{I}} \quad (**).$$

Supponiamo che la seguente asserzione sia in un grafo RDF G e sia G' l'ontologia delle asserzioni che è possibile inferire da G

```
ex:Bob ex:childOf ex:Eliza.
```

Allora

$$\begin{aligned} &\texttt{ex:Bob ex:childOf ex:Eliza} \in G && \Longrightarrow (**)\\ &\texttt{rdf:Property(ex:childOf)} \in G' && \Longrightarrow (*)\\ &\texttt{ex:childOf rdf:type rdf:Property} \in G' \end{aligned}$$

La classe `rdf:Property`

Ogni proprietà P presente in un grafo RDF deve essere una istanza della classe `rdf:Property`

$$\mathcal{I} \models a\, P\, b \iff [a^{\mathcal{I}}, b^{\mathcal{I}}] \in P^{\mathcal{I}} \wedge P^{\mathcal{I}} \in (\texttt{rdf:Property})^{\mathcal{I}} \quad (**).$$

Supponiamo che la seguente asserzione sia in un grafo RDF G e sia G' l'ontologia delle asserzioni che è possibile inferire da G

`ex:Bob ex:childOf ex:Eliza.`

Allora

$$\begin{aligned} &\texttt{ex:Bob ex:childOf ex:Eliza} \in G && \Longrightarrow (**)\\ &\texttt{rdf:Property(ex:childOf)} \in G' && \Longrightarrow (*)\\ &\texttt{ex:childOf rdf:type rdf:Property} \in G' \end{aligned}$$

Proprietà e Class in RDF

Si noti che le proprietà e le classi in RDF sono trattate anche come *individui*. In altre parole, nel contesto RDF

$$N_P \subseteq N_I,$$

$$\texttt{rdf:Property} \in N_C \cap N_I.$$

Operazioni di questo tipo sono in genere *rischiose* in termini di indecidibilità (vedi OWL Full).

In genere, la natura di individuo di una proprietà classi viene utilizzata solo per *annotare* le proprietà con commenti ed etichette esplicative oppure per descrivere i vincoli, come vedremo più avanti.

Serializzazione di grafi RDF

Il processo di serializzazione di un grafo RDF permette di scrivere un grafo su un file o di inviarlo in rete.

La serializzazione di un grafo RDF genera una stringa di testo in una delle seguenti *sintassi concrete*:

- NTriples
- Turle
- RDF XML
- RDFa
- ...

La Sintassi RDF N-Triples

Un grafo RDF serializzato secondo la sintassi N-Triples è una sequenza di righe con la seguente sintassi

$$\begin{array}{lcl} r & := & < IRI > \quad < IRI > \quad o. \\ o & := & < IRI > | \text{“STRING”} | \text{“STRING”}\hat{}\hat{}\texttt{<IRI>} \end{array}$$

dove *IRI* sono delle IRI e *STRING* stringhe UNICODE.

Intuitivamente, ogni riga di un file N-Triples rappresenta una property assertion RDF. Nel seguito un esempio di grafo RDF espresso in N-Triples.

```
<http://ex.org/Alice> <http://ex.org/childOf> <http://ex.org/Eliza> .
<http://ex.org/Bob> <http://ex.org/childOf> <http://ex.org/Eliza> .
<http://ex.org/Bob> <http://ex.org/childOf> <http://ex.org/Eliza> .
<http://ex.org/Eliza> <http://ex.org/fullName> "Eliza Smith" .
<http://ex.org/Eliza> <http://ex.org/age>
                        "63"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger> .
```

La Sintassi RDF Turle - Predicate List

Turtle estende N-Triples con alcune funzionalità.

Predicate List - Usando il simbolo ";" al posto di "." al termine di alcune righe è possibile specificare coppie [predicato, oggetto] che si riferiscono allo stesso soggetto. Ad esempio, le seguenti righe (N-Triples)

```
<http://ex.org/Eliza> <http://ex.org/childOf> <http://ex.org/Giorgia> .
<http://ex.org/Eliza> <http://ex.org/childOf> <http://ex.org/Francis> .
<http://ex.org/Eliza> <http://ex.org/fullName> "Eliza Smith" .
<http://ex.org/Eliza> <http://ex.org/age>
                        "63"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger> .
```

possono essere espresse in Turtle come segue:

```
<http://ex.org/Eliza> <http://ex.org/childOf> <http://ex.org/Giorgia> ;
                      <http://ex.org/childOf> <http://ex.org/Francis> ;
                      <http://ex.org/fullName> "Eliza Smith" ;
                      <http://ex.org/age>
                         "63"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger> .
```

La Sintassi RDF Turle - Object List

Analogamente, usando simbolo "," al posto di "." al termine di alcune righe è possibile specificare molteplici *oggetti* che si riferiscono alla stessa coppia [soggetto, predicato] Ad esempio

```
<http://ex.org/Eliza> <http://ex.org/childOf> <http://ex.org/Giorgia> .
<http://ex.org/Eliza> <http://ex.org/childOf> <http://ex.org/Francis> .
```

possono essere abbreviate con

```
<http://ex.org/Eliza> <http://ex.org/childOf> <http://ex.org/Giorgia> ,
                                              <http://ex.org/Francis> .
```

La Sintassi RDF Turle - Prefisso base

Con la parola chiave base è possibile specificare un *prefisso base* che verrà utilizzato per risolvere le IRI incomplete presenti nel grafo RDF.

```
BASE <http://ex.org/> .

<Eliza> <childOf>  <Giorgia> ,
                   <Francis> .
<Eliza> <fullName> "Eliza Smith" ;
        <age> "63"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger> .
```

La Sintassi RDF Turle - Prefissi

È possibile inoltre dichiarare anche altri *prefissi* da utilizzare per abbreviare le IRI.

```
BASE <http://ex.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> .

<Eliza>   <rdf:type> <Person> .
<Giorgia> <rdf:type> <Person> .
<Francis> <rdf:type> <Person> .
<Eliza>   <childOf>  <Giorgia> ,
                     <Francis> .
<Eliza>   <fullName> "Eliza Smith" ;
          <age>      "63"^^<xsd:nonNegativeInteger> .
```

La Sintassi RDF Turle - `rdf:type`

Il simbolo 'a' può essere usato al posto del predicato `rdf:type`.

```
BASE <http://ex.org/> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> .

<Eliza> a <Person> .
<Giorgia> a <Person> .
<Francis> a <Person> .
<Eliza> <childOf> <Giorgia> ,
                  <Francis> .
<Eliza>  <fullName> "Eliza Smith" ;
         <age> "63"^^<xsd:nonNegativeInteger> .
```

La Sintassi RDF Turle - Letterali

Turtle definisce alcune abbreviazioni anche per i letterali, che permettono di non indicare esplicitamente il data type. Ad esempio:

```
BASE <http://ex.org/>

<http://en.wikipedia.org/wiki/Helium>
    :atomicNumber 2 ;               # xsd:integer
    :atomicMass 4.002602 ;          # xsd:decimal
    :specificGravity 1.663E-4 .     # xsd:double
```

Per un elenco completo delle abbreviazioni disponibili in Turtle per i letterali fare riferimento a `http://www.w3.org/TR/2014/REC-turtle-20140225/#literals` .

La Sintassi RDF XML

La sintassi *RDF XML* permette di serializzare un grafo RDF in un documento XML. L'elemento radice di questo documento è di tipo `rdf:RDF`.

Gli elementi figli della radice sono di tipo `rdf:Description` e rappresentano *nodi* del grafo RDF (individui). La IRI associata al nodo viene specificata con l'attributo `rdf:about`. Nel caso in cui non sia presente l'attributo `rdf:about` siamo in presenza di un blank node.

I figli di ogni elemento di tipo `rdf:Description` sono le proprietà con le quali sono etichettati gli archi uscenti dal nodo che stiamo rappresentando.

A loro volta, tali elementi rappresentanti gli archi uscenti da un nodo possono avere come figli

- altri elementi di tipo `rdf:Description`, nel caso in cui l'oggetto della asserzione sia un individuo,
- degli elementi con un datatype come tipo, nel caso in cui l'oggetto della asserzione sia un letterale,
- oppure possono avere un attributo `rdf:resource` che ha come valore la IRI associata all'individuo oggetto dell'asserzione.

La Sintassi RDF XML - Property assertions

In altre parole, una property assertion del tipo `iriSubj iriProp iriObj`, con *iriSubj*, *iriProp*, *iriObj* $\in$ *IRI* può essere serializzata in RDF XML con il seguente elemento

```
<rdf:Description rdf:about="iriSubj">
  <iriProp>
    <rdf:Description rdf:about="iriObj" />
  <iriProp>
</rdf:Description>
```

oppure, utilizzando `rdf:resource`, con

```
<rdf:Description rdf:about="iriSubj">
  <iriProp rdf:resource="iriObj" />
</rdf:Description>
```

La Sintassi RDF XML - Multiple Property assertions

Inoltre, property assertions con lo stesso oggetto possono essere raggruppate all'interno di un unico elemento di tipo `rdf:Description`. Vediamo ad esempio una possibile serializzazione in RDF XML di due asserzioni con lo stesso soggetto.

*iriSubj iriProp*1 *iriObj*1 ,
*iriSubj iriProp*2 *iriObj*2

```
<rdf:Description rdf:about="iriSubj">
  <iriProp1 rdf:resource="iriObj1" />
  <iriProp2 rdf:resource="iriObj1" />
</rdf:Description>
```

La Sintassi RDF XML - String literals

I letterali di tipo stringa specificati come oggetto di una properti assertions vanno inseriti come contenuto (CDATA) dell'elemento che identifica la proprietà all'interno degli elementi `rdf:Description`.
Ad esempio una asserzione del tipo seguente

`iriSubj iriProp` "string"

può essere serializzata come segue:

```
<rdf:Description rdf:about="iriSubj">
  <iriProp>string</iriProp>
</rdf:Description>
```

La Sintassi RDF XML - Datatype property

È possibile inoltre specificare esplicitamente il datatype di un letterale oggetto di una property assertion mediante l'attributo `rdf:datatype`.

`iriSubj iriProp` "lexForm" `^^iriType`

```
<rdf:Description rdf:about="iriSubj">
  <iriProp rdf:datatype="iriType">lexForm</iriProp>
</rdf:Description>
```

La Sintassi RDF XML - Namespace

Le abbreviazioni per i prefissi possono essere implementate attraverso i meccanismi forniti dalla specifica XML. Segue un esempio completo di grafo RDF serializzato in RDF/XML.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://ex.org/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

  <rdf:Description rdf:about="Giorgia">
    <rdf:type rdf:resource="http://ex.org/Person" />
  </rdf:Description>

  <rdf:Description rdf:about="Francis">
    <rdf:type rdf:resource="http://ex.org/Person" />
  </rdf:Description>

  <rdf:Description rdf:about="Eliza">
    <rdf:type rdf:resource="http://ex.org/Person" />
    <childOf>
      <rdf:Description rdf:about="Giorgia" />
      <rdf:Description rdf:about="Francis" />
    </childOf>
    <fullName>Eliza Smith</fullName>
    <age rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">63</age>
  </rdf:Description>
</rdf:RDF>
```

La Sintassi RDF XML - Typed Node Elements

Se un individuo appartiene ad una classe con una qualche IRI *uriClass*, ossia se una una tripla del tipo

uriInd `rdf:type` *uriClass*

compare nel grafo RDF, è possibile non usare `rdf:Description` per indicare l'individuo *uriInd*, ma al suo posto dichiarare nel documento XML un elemento di tipo *uriClass*.

Ad esempio, e seguenti due serializzazioni sono equivalenti.

```
<rdf:Description rdf:about="Eliza">
  <rdf:type rdf:resource="http://ex.org/Person" />
  <childOf rdf:resource="http://ex.org/Giorgia" />
  <childOf rdf:resource="http://ex.org/Francis" />
  <fullName>Eliza Smith</fullName>
  <age rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">63</age>
</rdf:Description>
```

```
<Person rdf:about="Eliza">
  <childOf rdf:resource="http://ex.org/Giorgia" />
  <childOf rdf:resource="http://ex.org/Francis" />
  <fullName>Eliza Smith</fullName>
  <age rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">63</age>
</Person>
```

RDF Schema

RDF Schema (in breve *RDFS*) estende RDF con ulteriori classi e proprietà per descrivere *vincoli semantici*.

In questa sede vedremo `rdfs:Class` $\in N_C$ e le proprietà `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`, `rdfs:label`, `rdfs:comment` dove il prefisso `rdfs` è una abbreviazione per il namespace

`http://www.w3.org/2000/01/rdf-schema#`

Classi RDFS

In RDFS è definita una classr `rdfs:Class` cui appartengono tutte le altre classi:

$$C \in N_C \Longrightarrow C^{\mathcal{I}} \in (\texttt{rdfs:Class})^{\mathcal{I}}$$

per ogni interpretazione RDFS $\mathcal{I}$.

Gerarchie di Classi e Proprietà

In RDFS è possibile definire gerarchie di classi.

$$\mathcal{I} \models C \texttt{ rdfs:subClassOf } D \iff \mathcal{I} \models C \sqsubseteq D$$

$$\mathcal{I} \models P \texttt{ rdfs:subPropertyOf } Q \iff \mathcal{I} \models P \sqsubseteq Q$$

per ogni $C, D \in N_C$, $P, Q \in N_P$ e per ogni interpretazione RDFS $\mathcal{I}$.

Vincoli di Dominio e Codominio

Si possono esprimere restrizioni sul dominio e codominio di una proprietà

$$\mathcal{I} \models P \texttt{ rdfs:domain } C \iff \mathcal{I} \models \mathsf{dom}(P) \sqsubseteq C$$

$$\mathcal{I} \models P \texttt{ rdfs:range } C \iff \mathcal{I} \models \mathsf{range}(P) \sqsubseteq C$$

per ogni $C \in N_C$, $P \in N_P$ e per ogni interpretazione RDFS $\mathcal{I}$.

Annotazioni

È inoltre possibile *annotare* classi e proprietà con delle descrizioni intuitive.

```
<rdfs:Class rdf:about="http://ex.org/Man">
  <rdfs:subClassOf rdf:resource="http://ex.org/Person" />
  <rdfs:label>Man</rdfs:label>
  <rdfs:comment>A male person.</rdfs:comment>
</rdf:Class>
```

Vocabolari

Nomi di classe e proprietà vengono raggruppati in *vocabolari* che trattano specifici domini di conoscenza (eg. organizzazioni, pubblica amministrazione, biologia, commercio, etc.).

Un vocabolario può contenere anche *vincoli* inerenti classi e le proprietà del vocabolario stesso.

L'utilizzo di vocabolari condivisi permette lo sviluppo di applicazioni riguardanti specifici domini di conoscenza, ma indipendenti dai dati e dai dataset.

Definizione di Vocabolario

Una definizione di vocabolario può essere la seguente:

$$V = (C, P, \Omega)$$

dove

1. C è un sottoinsieme finito di N_C,
2. P è un sottoinsieme finito di N_P,
3. Ω è un insieme finito di vincoli che coinvolgano solo nomi di classi in C e nomi di proprietà in P.

Il Vocabolario FOAF

Uno dei primi e più utilizzati vocabolari definiti nell'ambito del Web semantico è *Friend OF A Friend* (*FOAF*, vedi `http://foaf-project.org`).

> *FOAF is a project devoted to linking people and information using the Web.*

Il vocabolario è disponibile alla URL

http://xmlns.com/foaf/spec/index.rdf

Il namespace del vocabolario FOAF è `http://xmlns.com/foaf/0.1/`

`foaf-a-matic`[2] è uno strumento *giocattolo* che permette di realizzare facilmente una ontologia che usa il vocabolario FOAF.

[2] `http://www.ldodds.com/foaf/foaf-a-matic.html`

FOAF Core

In questa sede ci limiteremo solo alla parte *Core*.

Il vocabolario *Foaf Core* è definito come segue:

$$
\begin{aligned}
\texttt{FOAFCore} &=_{\text{Def}} (C_{foaf}, P_{foaf}, \Omega_{foaf}) \\
C_{foaf} &=_{\text{Def}} \{Agent, Person, Project, Organization, Group, Document, Image\} \\
P_{foaf} &=_{\text{Def}} \{name, title, img, depiction, depicts, familyName, givenName, knows, \\
&\quad based\_near, age, made, maker, primaryTopic, primaryTopicOf, member\} \\
\Omega_{foaf} &=_{\text{Def}} \{Person \sqsubseteq Agent, Group \sqsubseteq Agent, Organization \sqsubseteq Agent, \\
&\quad Image \sqsubseteq Document, \text{dom}(title) \sqsubseteq Document, \\
&\quad \text{range}(depiction) \sqsubseteq Image, img \sqsubseteq depiction, \text{dom}(img) \sqsubseteq Person, \\
&\quad \text{dom}(knows) \sqsubseteq Person, \text{range}(knows) \sqsubseteq Person, \ldots\}
\end{aligned}
$$

FOAF Core - classi

Nel seguito lasceremo ometteremo il prefisso delle URI, dando per scontato che sia quello del vocabolario FOAF.

Nel *core* del vocabolario FOAF vengono definte le seguenti classi:

- *Agent* - appartengono a questa classe tutte quelle entità in grado di compiere azioni (persone, gruppi, software, robot, ...);
- *Person* ($Person \sqsubseteq Agent$) - persone (vive o morte, reali o immaginarie);
- *Group* ($Group \sqsubseteq Agent$) - insiemi di agenti;
- *Organization* ($Organization \sqsubseteq Agent$) - insiemi di persone che rappresenta una *istituzione sociale* (azienda, associazione, ministero, ...);
- *Document* sono i documenti, nel senso comune del termine (atti, leggi, carte di identità, ...);
- *Image* - ($Image \sqsubseteq Document$) - i documenti che sono immagini, sia digitali che non;
- *Project* - un incontro collettivo di qualche tipo.

FOAF Core - Proprietà

Elenchiamo ora le proprietà nel FOAF Core:

- *name* - il nome di qualcosa;
- *title* - titolo onorifico di una person (Mr, Mrs, Ms, Dr. etc);
- *familyName* ($\mathrm{dom}(familyName) \sqsubseteq Person$) - il cognome di una persona;
- *givenName* - prima parte del nome completo di una persona;
- *img* ($\mathrm{dom}(img) \sqsubseteq Person$, $\mathrm{range}(img) \sqsubseteq Image$) - mette in relazione una persona con una immagine che la rappresenta;
- *based_near* ($\mathrm{dom}(based\_near) \sqsubseteq Spatial_T hing$, $\mathrm{range}(based_n ear) \sqsubseteq Spatial_T hing$) - indica che due cose sono *vicine* in termini spaziali;
- *age* ($\mathrm{dom}(age) \sqsubseteq Agent$) - l'età, espressa in numero di anni, di un agente;
- *knows* ($\mathrm{dom}(knows) \sqsubseteq Person$, $\mathrm{range}(knows) \sqsubseteq Person$) - indica che è avvenuta interazione di qualche tipo tra due persone;
- *primaryTopicOf* ($\mathrm{dom}(primaryTopicOf) \sqsubseteq Document$) - indica l'oggetto principale di un documento;
- *isPrimaryTopicOf* ($\mathrm{range}(isPrimaryTopicOf) \sqsubseteq Document$) - mette in relazione un oggetto con i documenti che lo rigurdano;
- *made* ($\mathrm{dom}(made) \sqsubseteq Agent$) - mette in relazione un agente con qualcosa che ha prodotto;
- *maker* ($\mathrm{range}(made) \sqsubseteq Agent$) - mette in relazione un oggetto con gli agenti che hanno contribuito a crearlo;
- *member* ($\mathrm{dom}(member) \sqsubseteq Group$, $\mathrm{range}(member) \sqsubseteq Agent$) - mette in relazione un gruppo con i suoi membri.

Descrizioni Intuitive degli Elementi dei Vocabolari

Le classi e le proprietà di un vocabolario vengono spesso fornite di una descrizione intuitiva nel documento che descrive il vocabolario. Ad esempio, le classi *Agent* e *Person* vengono descritte come segue in `http://xmlns.com/foaf/spec/`

> **Agent** - *The Agent class is the class of agents; things that do stuff. A well known sub-class is Person, representing people. Other kinds of agents include Organization and Group.*
> *The Agent class is useful in a few places in FOAF where Person would have been overly specific. For example, the IM chat ID properties such as jabberID are typically associated with people, but sometimes belong to software bots.*
>
> **Person** - *The Person class represents people. Something is a Person if it is a person. We don't nitpic about whether they're alive, dead, real, or imaginary. The Person class is a sub-class of the Agent class, since all people are considered 'agents' in FOAF.*

Descrizioni Rigorose degli Elementi dei Vocabolari

Tuttavia, già nei vincoli di un vocabolario si trovano indicazioni importanti sulla *semantica* dei nomi di classe e di proprietà del vocabolario stesso.

$$
\begin{aligned}
&\mathit{Person} \sqsubseteq \mathit{Agent}\\
&\mathrm{range}(\mathit{depiction}) \sqsubseteq \mathit{Image}\\
&\mathit{img} \sqsubseteq \mathit{depiction},\\
&\mathrm{dom}(\mathit{img}) \sqsubseteq \mathit{Person},\\
&\mathrm{dom}(\mathit{knows}) \sqsubseteq \mathit{Person},\\
&\mathrm{range}(\mathit{knows}) \sqsubseteq \mathit{Person},\\
&\ldots
\end{aligned}
$$