Data Semantics

Sommario

La *Data Semantcs* si occupa di comprendere il significato dei dati durante la scrittura di un programma. Nell'ambito dell'analisi è fondamentale nell'integrazione di più dataset e nella loro condivisione. Altra problematica che la *Data Semantics* si pone di risolvere è l'organizzazione di dati non strutturati in modo da facilitarne l'analisi.

Scopo del corso è costruire dei modelli per attribuire valore semantico ai dati, in modo tale da facilitarne la fruizione; si tenta inoltre di capire il ruolo della semantica nell'integrazione dei dataset. Le finalità del corso rientrano nell'ambito dei *big data*, nella costruzione di *knowledge graphs* (ovvero la costruzione di relazioni interne a un database) e la costruzione di sistemi di raccomandazione. Saranno inoltre analizzate alcune tecniche di *natural language processing* per costruire rappresentazioni a partire dai dati.
Le esercitazioni si occuperanno di interrogare *knowledge graphs* e integrare fonti di dati.

L'esame orale sarà accompagnato da un progetto software (effettuato in gruppo di - circa - 3 persone), di cui sarà fatta una presentazione orale; in alternativa al progetto è possibile scrivere un articolo di approfondimento su una tematica a scelta. La preparazione dovrà essere "ragionevolmente" dettagliata su tutti gli argomenti, e particolarmente approfondita sull'argomento del progetto.

Indice

Parte I
Grafi di conoscenza

Il termine è coniato da Google per indicare uno strumento usato dal suo motore di ricerca per fornire informazioni aggiuntive durante le ricerche. La struttura a grafo permette di essere processata facilmente da una macchina e allo stesso tempo garantisce un livello di astrazione soddisfacente; si possono anche effettuare query come in un database a grafo (Neo4j). Il modello a grafo permette inoltre una facile integrazione di sorgenti diverse rendendo naturali collegamenti e relazioni. Il web *semantico* ha costruito linguaggi e strumenti, approvati dal W3C, per definire, interrogare e fare inferenza su grafi di conoscenza; tuttavia nel mondo reale questi strumenti non hanno largo utilizzo. La costruzione di grafi di conoscenza è spesso effettuata a mano da una moltitudine di utenti (chiedendo recensioni tramite *app*).

Internet produce enormi quantità di dati diversi tra di loro, usati spesso per altri fini: la semantica dei dati si occupa di integrare grandi quantità (*data volume*) da diverse fonti di dati (*data variety*). Tali dati possono essere sfruttati nella costruzione di intelligenze artificiali, ovvero di programmi che eseguono task tipicamente umani con risultati simili. In particolare è così possibile effettuare compiti particolarmente ardui per un umano, come l'integrazione di serie storiche con fonti multiple appartenenti a domini vari o poco noti.

Esistono grafi di conoscenza *open*, quali DBpedia, Yago o Wikidata, ma alcune aziende private sviluppano il proprio grafo per facilitare l'attività imprenditoriale.

1 Spazio vettoriale.

In un grafo di conoscenza, query e documenti sono interpretabili come vettori (in un sistema tradizionale di information retrieval si agisce attraverso un modello matematico, il *il term document matrix* in cui ogni cella ci dice se le parole ricorrono o meno nel documento): è possibile calcolare misure di distanza come la *distanza euclidea* oppure calcolare misure di similarità come la *distanza coseno*.

$$\begin{aligned} sim(r,u) &= cos(\theta) \\ &= \frac{uq}{|u||q|} \end{aligned}$$

La rappresentazione vettoriale è alla base dell'analisi testuale. Mentre in passato si sfruttavano sistemi basati essenzialmente sulla presenza-assenza di una parola e quindi sui valori 0 e 1 oggi si usa un sistema di pesi più sofisticato per valutare diversamente l'importanza di una parola all'interno di un documento, quindi con valori *compresi* tra 0 e 1. Una parola è tanto più importante nel documento quante più ricorrenze ci sono nel documento e meno per quante ricorrenze ha negli altri documenti (in percentuale).

$$\begin{aligned} tf_{i,j} &= \frac{n_{i,j}}{|d_j|} \\ idf_i &= \log \frac{|D|}{|\{d : i \in d\}|} \\ tfidf_{i,j} &= tf_{i,j} \times idf_i \end{aligned}$$

Le entità, le loro proprietà e i collegamenti tra di loro sono iscritti nel grafo di conoscenza. I grafi di conoscenza sono basi di conoscenza che suppportano task di data analytics perchè possiedono un contesto ricco, servizi di collegamento e supportano accesso via web; inoltre rappresentano l'astrazione più importante per la data semantics Esistono linguaggi formali, definiti a inizio '900, che permettono di dichiarare relazioni tra entità ma che non sono leggibili da una macchina. Inoltre si possono usare assiomi logici per definire le ricorrenze nel testo.

2 *Linking Data.*

I dati sono spesso collegati in modo automatico grazie a programmi di apprendimento automatico. Una ricerca su internet non è fatta per documenti ma per contenuti: un motore di ricerca in passato offriva una serie di documenti senza offrire informazioni aggiuntive; oggi invece un motore di ricerca tenta direttamente di rispondere con *factual information* rilevanti nella ricerca attraverso delle schede. Per *informazione fattuale* si intende un'informazione interpretabile come vera o falsa (al contrario, alcuni dati quali immagini o suoni non sono interpretabili per veridicità). I fatti costituiscono un elemento centrale per l'analisi.

Le risposte fattuali sono personalizzate in base alla natura della ricerca, secondo criteri *data driven* (ovvero statistici). Informazioni di tipo diverso hanno caratteristiche diverse: si produce un grafo di conoscenza per gestire meglio entità di tipo diverso con caratteristiche peculiari diverse. Le *preview* dei contenuti sono generate chiedendo agli sviluppatori di inserire dei contenuti nel codice HTML che sono poi interpretati dal motore di ricerca.

I chatbot sono costruiti con l'ausilio di reti neurali e rappresentazioni del mondo reale.

3 Standard.

RDF (*Resource Description Framework*) è il linguaggio standrad per il web semantico, ovvero per *l'internet elaborabile dalle macchine*. L'unità base per rappresentare l'informazione è rappresentata da triple (affermazioni), ovvero grafi etichettati, identificati da URI (*Unique Resource Identifier*). Esempi di triple sono:

```
<Electric Piano, label, "Electric Piano"@en>
<Elton John, instrument, Electric Piano>
<Sails, artist, Elton John>
```

Le triple sono rappresentabili come un grafo diretto, aciclico ed etichettato:

```
     Elton John -[artist]- Sails
     /          \
  [instrument]   [artist]- Empty Sky
   /
 Electric Piano
    \
     [label]- "Electric Piano"@en
```

Alle triple si applicano degli identificativi globali (una sorta di chiave primaria SQL, gli URI, appunto): si usano generalmente identificativi web (abbreviati, detti prefissi), che specificano

come recuperare l'informazione (qualsiasi cosa a cui si può attribuire un valore costante è definibile come URI). Tutti gli URI sono risorse, e rappresentano l'ontologia del dominio.

Le triple sono strutturate con un **soggetto**, un **predicato** e un **oggetto** (Turtle syntax). Il soggetto è composto da un URI o da un *blank node* (ovvero costanti *locali*), il predicato può essere composto solo da un URI mentre gli oggetti possono essere URI, *blank nodes* oppure letterali, che possono essere *plain literal*, ovvero letterali puri, oppure *typed literal*, a cui è attribuito un tipo (stringhe, date o booleani) per permettere operazioni. I *blank node* sono nodi anonimi per consentire una buona costruzione del grafo.

3.1 Pubblicazione.

I *linked data* riassumono delle best practices per pubblicare e unire dati provenienti dal web:

- usare URI come nomi delle cose;
- usare indirizzi HTTP come URI;
- inserire informazioni utili su un URI;
- includere link ad altri URI.

Sono solamente una serie di principi, non rendono i dati *open* (*Linked Open Data*).

Altro approccio per pubblicare i dati sul web è usare l'*annotazione semantica*: si inseriscono annotazioni (tag) in una pagina web che specifichino il significato del contenuto o altri meta-dati per permettere e facilitare l'elaborazione dei contenuti da parte di agenti intelligenti. La sintassi è simile a XML: RDF in precedenza usava il formato XML, non Turtle, ma è stato abbandonato per la sua struttura ad albero in favore di una struttura a grafo. L'approccio è tornato in auge grazie ai *crawler*: dei motori di ricerca che annotano le pagine in base al loro significato semantico[1].

Non tutti i formati sono compatibili con RDF (Microdata e hCard non lo sono) mentre altri sì (RDFa, JSON-LD); nonostante questo non ci sono differenze significative tra i vari formati (il modello è il medesimo).

hCard e vCard sono basati su HTML, ponendo delle convenzioni su come gestire le informazioni.
I microdati usano tag speciali introdotti in HTML5 per definire *itemscope*, *itemtype* e *itempropriety*: si definisce (*itemscope*) un oggetto di un certo tipo (*itemtype*) con determinate proprietà (*itemprop*), che in RDF si inseriscono tramite *blank node*. I microdati sono parti di codice integrate dentro al testo, visualizzabili solamente nel codice sorgente. Si usano dei tipi definiti da vocabolari[2], utilizzabili anche in RDF, adottati come standard.
RDFa specifica all'inizio il vocabolario adottato, mentre il codice rimane scritto in HTML.
JSON-LD (JSON *Linked Data*) integra un file `.json` (in un tag `<script>`) in una pagina HTML per attribuire significati semantici. Non si ha bisogno di un parser apposito, e le triple sono facilmente ricostruibili a partire dalla pagina web. I motori di ricerca, processando in modo automatico, mostrano il contenuto in base al significato semantico.

[1] `https://webdatacommons.org` è un archivio di crawl trovati sul web.

[2] `https://schema.org/` è un dizionario che definisce in modo non eccessivamente prescrittivo oggetti e proprietà.

4 SPARQL.

Introdotto come linguaggio per interrogare *linked data*, è un linguaggio simile a SQL caratterizzato (dalla versione 1.1) da quattro elementi:

- SPARQL Query;
- SPARQL Algebra;
- SPARQL Update;
- SPARQL Protocol.

Permette di interrogare (query), modificare (update) e integrare (federated) database composti da triple. Il principio di SPARQL è quello di *pattern matching*, ovvero le query descrivono sottografi del grafo principale e quelli che matchano rappresentano i risultati. Si definisce il concetto di *triple pattern*: si sostituiscono uno o più elementi di una tripla con una variabile.

```
dbpedia:The_Beatles foaf:name "The Beatles" . # importante il punto alla fine
dbpedia:The_Beatles foaf:name ?album .
?album mo:track ?track .
?album ?p ?o .
```

Si possono così costruire query strutturate.

```
# prologo: si definiscono le fonti
PREFIX dbpedia:<http://dbpedia.org/resource/>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
...

# query di tipo SELECT
# ASK | SELECT | DESCRIBE | CONSTRUCT
  SELECT ?album
    FROM <http://musicbrainz.org/20130302/>
   WHERE {
           # il cuore della query:
           # il titolo di tutte le tracce degli album dei Beatles
           dbpedia:The_Beatles foaf:made ?album .
           ?album  a(shortcut per rdf:type) mo:Record;
           dc:title ?title .
         }
ORDER BY ?title
```

`ASK` ritorna `true`/`false` se esiste una soluzione alla query:

```
  ASK
WHERE {
        dbpedia:The_Beatles mo:member .
        dbpedia:Paul_McCartney .
```

```
      }

=> true
```

`SELECT` ritorna una lista di elementi che hanno corrispondenza nel grafo:

```
SELECT ?album_name ?track_title
       ?date       ?album       .
 WHERE {
         dbpedia:The_Beatles foaf:name ?album .             # tutti gli album
                                                            # dei Beatles
         ?album              dc:title  ?album;
         mo:track ?track .                                  # tutte le tracce
                                                            # degli album
         ?track            dc:title  ?track_title;
         mo:duration ?duration .
         FILTER (?duration > 300000 && ?duration < 400000)  # filtra per durata
       }
```

`CONSTRUCT` è simile a `SELECT` ma ritorna un grafo e non una lista:

```
CONSTRUCT {
            # riscrive con nuova terminologia
            ?album dc:creator dbpedia:The_Beatles
          }
    WHERE {
            # esegue la query
            dbpedia:The_Beatles foaf:made   ?album;
            ?album              mo:track    ?track .
          }
```

Le basi di conoscenza sono interrogate grazie ai vocabolari (anche detti *ontologie*), che permettono quindi la costruzione di query.

I dataset sono integrati grazie a predicati (`same as`) che indicano quali entità si riferiscono allo stesso oggetto reale nelle varie fonti.

SPARQL 1.1 permette di integrare un sistema di ragionamento automatico per l'uguaglianza delle entità (introduce il concetto di *entailment*).

5 Vocabolari (o ontologie).

In RDF si ha un modello dei dati che prevede di utilizzare URI per rappresentare sia le risorse sia le proprietà. L'uso di URI per specificare proprietà e relazioni definiscono separatamente l'insieme di proprietà e tipi usati per descrivere un dominio: si definisce così un vocabolario per descrivere i dati, che permettono una standardizzazione della tipizzazione del dato. Un esempio è il predicato `rdf:type` che introduce un vocabolario che permette di catalogare risorse definendo i tipi di variabile. Si definiscono quindi ontologie comuni sfruttabili da più *data provider* per uniformare la rappresentazione dei dati. A tutti gli effetti un vocabolario è riconducibile allo schema dei RDBMS.

Per definire un vocabolario si definiscono prima le classi (ovvero i tipi di dato) con proprietà, eventualmente diverse (ovvero usando certe proprietà solo su certe classi).

Uno dei vocabolari più importanti (perché introdotto da tempo) è **FOAF** (*Friend of a Friend*); si ricordano anche **DC** (*Dubling Core*), che specifica una serie di predicati per rappresentare i documenti come *titolo, autore* ad esempio, e **SKOS** (*Simple Knowledge Organizating System*). Più vocabolari possono riferirsi allo stesso oggetto e per definizione non è lecito unificare i vocabolari in quanto dovrebbero adattarsi alla specificità del dominio. C'è quindi una certa libertà nel descrivere gli oggetti perchè ognuno può usare vocabolari differenti, portando ad un certo equilibrio nell'entropia del semantic web. Alcuni dei più grandi provider internet hanno però unificato vocabolari col progetto **schema.org**, molto utilizzato nel web e in continua evoluzione per dati strutturati. Sul sito di **schema.org** sono raccolte tutte le classi disponibili in questo vocabolario e per ogni classe restituisce la specifica delle proprietà per descriverla, oltre a fornirci anche gli *expected type* per i valori di queste proprietà; un concetto, questo, molto importante nella definizione dei vocabolari. Dipendentemente dal task occorre sempre trovare il vocabolario più semplice per coprire tutti i nostri bisogni ed eventualmente, nel caso si rivelasse insufficiente, espanderlo per i nostri bisogni, piuttosto che crearne uno nuovo dal nulla. Si possono unire più vocabolari grazie a collegamenti, a livello di istanza o di schema.

Un computer ha bisogno di definire un sistema per attribuire un significato al significante. Si possono definire tre[3] sistemi per definire un'ontologia, ovvero un sistema per definire gli oggetti esistenti.

L'uso di sistemi diversi rende complicata l'integrazione di più fonti.

5.1 Tesauri.

Tra i diversi tipi di ontologie il tesauro è il più obsoleto. Il tesauro può essere definito anche come ontologia lessicale, ovvero i significati delle parole sono esplicitati in termini di relazioni lessicali tra queste. Generalmente definisce sostantivi (semplici o composti), aggettivi e verbi.

Le relazioni lessicali sono:

- **sinonimia**: più parole con significati simili;
- **antinomia**: più parole hanno significato opposto;
- **polisemia**: più significati della stessa parola;
- **iponimia** (e iperonimia): quando un significato rappresenta un sottoinsieme dell'altro (casa - costruzione);
- **associazione**: più parole strettamente legate tra di loro.

Tuttavia esistono dei problemi: i sinonimi non sono mai perfettamente interscambiabili e l'iponimia (o iperonimia) sottintende molte sfumature.

WordNet è un vasto database lessicale sviluppato dagli anni '90 che rappresenta i significati (o concetti) in una struttura ad albero. La polisemia è gestita definendo più entità diverse con lo stesso nome. Data una parola, si ha quindi un insieme finito di significati distinti tra di loro.

[3]Esiste un quarto metodo che sarà approfondito più avanti.

5.2 Tassonomia.

Si definisce un grafo di classificazione gerarchica dei concetti, generalmente con struttura ad albero. Le tassonomie sono strutture usate in informatica e nello specifico in Data Science, prese in prestito dalla biologia (per classificare le forme viventi), in cui tuttavia sono presenti dibattiti. La valenza può anche non essere perfetta ma ha grande importanza in ambito scientifico ed economico. Si usa una forma ad albero (per facilitare la condivisione tra varie fonti) i cui nodi sono concetti (ogni nodo può avere un solo padre e uno o più figli). Il significato è spiegato in classi, superclassi e sottoclassi (relazione di tipo `is-a`) grazie al meccanismo dell'ereditarietà: i nodi più specializzati ereditano le proprietà di quelli più generali. È necessario specificare anche quale criterio usare per effettuare la divisione dell'albero. Si può anche effettuare una ripartizione dell'oggetto dividendolo nei sotto-oggetti specifici che lo compongono (mereologia), nell'ambito di una relazione `part-of`, ovvero una relazione transitiva e riflessiva .

5.3 Ontologie assiomatiche.

Il significato è attribuito attraverso una serie di assiomi logici per specificare le definizioni dei termini utilizzati, con riferimento alle implicazioni. Le caratteristiche fondamentali di un ontologia sono che essa sia concettuale, esplicita, formale e, caratteristica più importante, *condivisa*: è infatti fondamentale che le descrizioni siano condivise tra individui di un gruppo (**schema.org** è un esempiio di questo principio) Si definisce quindi un linguaggio L di assiomi logici con una semantica formale che determina il significato in modo univoco.

6 Inferenza nell'ontologia.

Definendo un oggetto, si hanno sempre delle implicazioni: è necessario fare inferenza, ovvero si procede con un procedimento deduttivo. L'inferenza (un meccanismo fondamentale della semantica) nasce infatti dal ragionamento su delle premesse, con l'obiettivo di estrarre conoscenza vera da premesse vere, effettuando una generalizzazione con un certo margine di errore. Dal principio dell'informatica, l'uso della logica (proposizionale e del primo ordine) permette la deduzione da parte di calcolatori. Oltre all'induzione classica, esistono altre pratiche per fare inferenza. Le implicazioni ricavate sulla base di ciò che è definito nel dizionario, dal punto di vista inferenziale, sono la semantica reale delle informazioni. L'inferenza ha come obiettivi di ridurre la complessità del sistema trovando un modo per ottenere informazioni velocemente e per aumentare la flessibilità dello schema per poter essere aggiornato in tempo reale con le modifiche dell'ambiente.

6.1 Schema dell'ontologia.

Lo schema prescrive come deve essere organizzato il dato per entrare nel database; tuttavia i dati sono flessibili per loro natura, dunque l'operazione non è immediata. È necessario definire uno schema per dare una struttura ai dati: non è lecito usare nell'approccio semantico un database NoSQL troppo flessibile. Dall'altro lato rispetto all'approccio relazionale, lo schema semantico è più flessibile: SQL non permette di modificare la struttura una volta dichiarata, usa cioè un approccio *prescrittivo*. Quello del semantic web è invece un approccio *deduttivo*

(*validazione indiretta*): ovvero non ci dice come devono essere strutturati i dati per essere inseriti, ma quali informazioni possiamo estrarne (validazione indiretta); in sintesi tiene conto del fatto che i dati, nel tempo, possano cambiare. Lo schema quindi è flessibile e disaccoppia lo schema dai dati (ovvero lo schema può essere modificato successivamente e ciò potrebbe provocare incoerenze nel breve periodo).

Potendo esserci contraddizione nei dati, si sono costruiti linguaggi come RDF Schapes e SHACL che permettono una validazione dei dati attraverso la definizione di vincoli.

6.2 *Reasoning.*

I linguaggi RDFS e OWL permettono l'operazione di *reasoning*, ovvero l'inferenza di nuova conoscenza a partire da più antologie esistenti. Oltre ad avere sistemi deduttivi, la logica proposizionale garantisce che l'algoritmo termini (è cioè *decidibile*, ovvero date le premesse è sempre possibile stabilire se una FBF - Formula Ben Formata - è vera o meno), mentre la logica del primo ordine è semi-decidibile. Anche con una procedura che termina e confermi la verità di una formula però non è utile in pratica: il tempo impiegato potrebbe essere particolarmente lungo e quindi rendere inutile il programma. In caso di linguaggi non decidibili, bisogna semplificare il linguaggio per trovare un compromesso tra tempo impiegato ed espressività: ci sono più linguaggi perchè il compromesso è trovato in modo diverso. Inoltre esistono altri linguaggi *a regole*, ovvero linguaggi non puramente logici, ma che vi si ispirano, utilizzabili per fare deduzioni su RDF che sono più efficienti.

6.2.1 RDF Schema.

RDFS costruisce un *reticolo* che parte da un nodo e garantisce ereditarietà (anche multipla) delle classi: è costruito un *ordine parziale*. RDFS introduce implicazioni sul tipo di dato: si introduce un approccio secondo cui il mondo è composto da *classi* e *individui*; quindi si hanno insiemi ed elementi degli insiemi. Una classe è composta da individui, i quali tramite *proprietà* entrano in relazione tra di loro; inoltre le relazioni tra classi comportano vincoli sulle relazioni tra individui. Questo approccio è coerente con il modello relazionale, e caratterizza in modo molto forte la semantica dei dati strutturati. Il concetto di classe vincola l'uso del tipo: una classe non può essere del tipo di un'altra.

I letterali sono divisi esplicitamente dagli URI.

Con RDFS si possono rappresentare e vincolare relazioni tra classi, organizzandole in gerarchie: si organizzano così tassonomie che garantiscono nativamente inferenze semplici (con proprietà transitiva). Anche le proprietà possono essere organizzate gerarchicamente, con proprietà che derivano da altre (`figlioDi` è sotto-proprietà di `parenteDi`).

La semantica dei predicati è definita da regole, che determinano l'ereditarietà (classi derivate), la transitività di classi e relazioni e simili.

I predicati `rdfs:domain` e `rdfs:range` indicano che tutti gli oggetti di una classe appartengono anche a un'altra classe. Questi due predicati non sono usati in `schema.org` perché troppo rigidi. Lo schema non può dedurre contraddizioni, dunque.

Alcuni esempi di quanto detto sono:

```
se [x sottoclasse di y]
e  [a type x]
allora [a type y] (i tipi si ereditano)
```

```
se [x sottoclasse di y]
e  [y sottoclasse di z]
allora [x sottoclasse di z] (proprietà transitiva)

se [x a y]
e  [a sottoproprietà di b]
allora [x b y] (i precedenti esempi
                valgono anche per le proprietà)

se [a sottoproprietà di b]
e  [b sottoproprietà di c]
allora [a sottoproprietà di c]

se [x a y]
e  [a domain z]
allora [x type z]
(tutti quei soggetti che hanno una proprietà parte
di un dominio, sono dello stesso tipo del dominio)

se [x a u]
e [a range z]
allora [u type z]
(stesso ragionamento del dominio ma il range vale
per l'oggetto, non per il soggetto.)
```

Nonostante tutti i pregi, RDFS presenta alcuni svantaggi: non ha restrizioni locali all'appartenenza di domini, non ha vincoli di cardinalità e le classi non possono essere combinate. OWL è stato introdotto per colmare queste lacune.

6.2.2 SPARQL 1.1.

La versione 1.1 introduce gli *Entailment Regimes*: procedure che calcolano tutte le implicazioni degli assiomi; uno dei sistemi più noti è Virtuoso. I risultati possono essere esportati come `.xml`, `.csv` o `.tsv`.

6.2.3 OWL.

È un motore inferenziale composto da più sotto-linguaggi, con l'obiettivo di ridurre la confusione lasciata da RDFS, tuttavia ne aumenta la complessità computazionale. È un linguaggio *formale* (è un linguaggio logico processabile da elaboratori), *esplicito* (non ambiguo) e *condiviso*. OWL2 è diviso in tre progetti:

- OWL 2 EL: limitato a classificazioni di base, ma la complessità è polinomiale;
- OWL 2 QL: usa una sintassi simile a SQL;
- OWL 2 RL: è usato per essere implementato efficientemente in sistemi *rule-based*.

È frequente definire ontologie senza usare tutta la capacità espressiva di OWL.

Rispetto a RDFS si introduce l'equivalenza tra classi con `owl:equivalentClass` e tra individui con `owl:sameAs`, tuttavia il loro uso ha delle implicazioni semantiche (riflessività, transitività e simmetria); inoltre l'equivalenza può essere fatta anche a livello di proprietà con `owl:equivalentProperty`. Si può anche specificare la disgiunzione tra due elementi con `owl:differentFrom`, in opposizione alla UNI (*Unique Name Assumption*), secondo cui oggetti con nomi diversi sono distinti. Le proprietà possono essere disgiunte con `owl:propertyDisjointWith`.

Le proprietà di OWL sono distinte in due categorie: **ObjectProperties** (la proprietà è una risorsa) o **DatatypeProperties** (la proprietà è un letterale); questa distinzione facilita la costruzione di grafi specificando quali proprietà sono utilizzabili per attraversare il grafo.

La relazione tra classi e sottoclassi è, matematicamente parlando, un ordinamento parziale: si può costruire un reticolo finito (quindi con limite superiore e inferiore, rispettivamente uguali all'insieme *universo* e all'insieme *vuoto*).

Una relazione in OWL può possedere determinate proprietà:

- **Simmetria**;
- **Transitività**;
- **Transitività inversa**;
- **Funzionalità**: ovvero si ha al più un valore per un argomento permettendo l'uso di identificativi e di funzioni inverse;
- **Funzionalità inversa**: quest'ultima proprietà è fondamentale nelle fasi di *Record Linkage* o *Deduplication*.

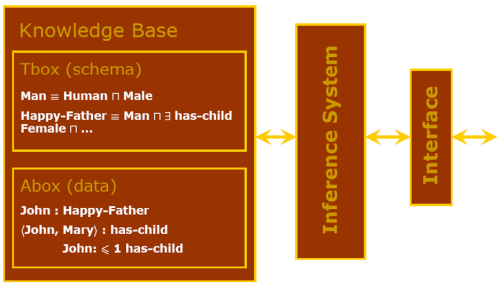
Un concetto si dice:

- *Soddisfacibile* se esiste almeno una sua interpretazione;
- *Sussunto* a un altro concetto se qualsiasi interpretazione del primo concetto è sottoinsieme di un interpretazione del secondo;
- Due concetti si dicono *equivalenti* se l'interpretazione del primo concetto è sempre uguale a quella del secondo in tutte le interpretazioni della teoria, al contrario si dicono *disgiunti*.

Il concetto di interpretazione in logica quindi è puramente teorico: è lo strumento matematico usato per attribuire un significato ai dati; l'inferenza permette di attribuire significati aggiuntivi indipendentemente dalla semantica iniziale. È possibile utilizzare il *reasoner* per verificare la correttezza delle relazioni. La conoscenza ha senso se è soddisfacibile: una teoria è incosistente se non ha interpretazioni che la rendono vera; caso analogo si verifica quando una classe è sussunta all'insieme vuoto: in tal caso non ha interpretazioni possibili. Dal punto di vista pragmatico, con un *reasoner* si verifica la soddisfacibilità della teoria, si inferiscono gerarchie e nuove relazioni e ancora il tipo degli individui.

7 Logiche descrittive.

Sono una famiglia di logiche usate per rappresentare il grafo di conoscenza; più una logica è complessa ed espressiva maggiori sono il tempo di esecuzione e la complessità. Sono state introdotte per lo studio del linguaggio naturale prima del *deep learning*. OWL DL (*OWL Descriptive Logic*) si basa su SHIQ (è equivalente a SHOINDn). Una logica è composta da una TBox (*Terminological Box*), ovvero l'ontologia della logica, e una ABox (*Assertional Box*), che comprende i dati veri e propri. Nelle logiche descrittive la difficoltà è costruire classi complesse (coi relativi costruttori).



Una classe può quindi essere definita basandosi su più classi tramite una serie di operatori logici. La semantica dunque si definisce ricorsivamente dagli elementi più semplici ai più complicati.

Constructor	DL Syntax	Example	Modal Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male	$C_1 \wedge \ldots \wedge C_n$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer	$C_1 \vee \ldots \vee C_n$
complementOf	$\neg C$	¬Male	$\neg C$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} ⊔ {mary}	$x_1 \vee \ldots \vee x_n$
allValuesFrom	$\forall P.C$	∀hasChild.Doctor	$[P]C$
someValuesFrom	$\exists P.C$	∃hasChild.Lawyer	$\langle P \rangle C$
maxCardinality	$\leqslant nP$	⩽1hasChild	$[P]_{n+1}$
minCardinality	$\geqslant nP$	⩾2hasChild	$\langle P \rangle_n$

- C is a concept (class); P is a role (property); x is an individual name
- XMLS datatypes as well as classes in ∀P.C and ∃P.C
  - Restricted form of DL concrete domains

L'*interpretazione* dei costrutti è insiemistica:

- Interpretation function $\cdot^{\mathcal{I}}$ extends to concept (and role) expressions in the obvious way, i.e.:

$$
\begin{array}{lcl}
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
\{x\}^{\mathcal{I}} & = & \{x^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} & = & \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} & = & \{x \mid \forall y.(x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\} \\
(\leqslant nR)^{\mathcal{I}} & = & \{x \mid \#\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \leqslant n\} \\
(\geqslant nR)^{\mathcal{I}} & = & \{x \mid \#\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \geqslant n\} \\
(R^{-})^{\mathcal{I}} & = & \{(x, y) \mid (y, x) \in R^{\mathcal{I}}\}
\end{array}
$$

Gli assiomi usati in queste logiche sono estremamente semplici ma si applicano ugualmente anche a classi più complesse. Avendo infatti complicato le classi possiamo ottenere assiomi più *semplici*:

OWL Syntax	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor

- Obvious FO/Modal Logic equivalences
  - E.g., DL: $C \sqsubseteq D$ FOL: $\forall x.C(x) \rightarrow D(x)$ ML: $C \rightarrow D$
- Often distinguish two kinds of Tbox axioms
  - "Definitions" $C \sqsubseteq D$ or $C \equiv D$ where C is a concept name
  - General Concept Inclusion axioms (GCIs) where C may be complex

Possiamo definire un interpretazione anche degli assiomi. L'interpretazione I di un assioma soddisfa l'assioma A ($I \models A$) solamente se sono rispettate alcune regole: con una teoria logica composta da assiomi, se ne possono avere un'infinità di interpretazioni ma ogni assioma riduce l'insieme di interpretazioni possibili. Un'ontologia ha esattamente questo compito: ridurre il numero di interpretazioni e permettere l'inferenza.

- An interpretation $\mathcal{I}$ satisfies (models) an axiom A ($\mathcal{I} \models$ A):
  - $\mathcal{I} \models C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ $\quad$ $\mathcal{I} \models C \equiv D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$
  - $\mathcal{I} \models R \sqsubseteq S$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ $\quad$ $\mathcal{I} \models R \equiv S$ iff $R^{\mathcal{I}} = S^{\mathcal{I}}$
  - $\mathcal{I} \models x \in D$ iff $x^{\mathcal{I}} \in D^{\mathcal{I}}$
  - $\mathcal{I} \models \langle x,y \rangle \in R$ iff $(x^{\mathcal{I}},y^{\mathcal{I}}) \in R^{\mathcal{I}}$
  - $\mathcal{I} \models R^+ \sqsubseteq R$ iff $(R^{\mathcal{I}})^+ \subseteq R^{\mathcal{I}}$
- $\mathcal{I}$ satisfies a Tbox $\mathcal{T}$ ($\mathcal{I} \models \mathcal{T}$) iff $\mathcal{I}$ satisfies every axiom A in $\mathcal{T}$
- $\mathcal{I}$ satisfies an Abox $\mathcal{A}$ ($\mathcal{I} \models \mathcal{A}$) iff $\mathcal{I}$ satisfies every axiom A in $\mathcal{A}$
- $\mathcal{I}$ satisfies an KB $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) iff $\mathcal{I}$ satisfies both $\mathcal{T}$ and $\mathcal{A}$

8 *Search Engine* semantici.

Un *search engine* semantico sfrutta una base di conoscenza per effettuare delle query più mirate. Per la gestione di pattern di architettura ci sono tre possibili approcci:

- **Crawling**: i dati sono memorizzati in un database locale;
- **On-the-fly deferencing**: gli URI sono deferenziati quando il programma richiede i dati;
- **Federated query**: ogni richiesta è delegata ad un servizio diverso.

Anche per il *reasoner* ci sono strategie diverse.

9 *Semantic Framework.*

Si usano dei framework per archiviare i dati in grafi per la facilità della gestione delle relazioni (si interroga per ottenere nuova conoscenza):

- **RDF4J** è un framework per Java per gestire, archiviare (in diversi modi) e interrogare documenti RDF, organizzando tramite un paradigma ad oggetti;

- **Aoache Jena** è un'api per RDF, offre un reasoner organizzando i dati ad oggetti;
- Un **triplestore** archivia le triple e le filtra per elementi fissati ed è fatto per ricevere query SPARQL. Ci permette di fare inferenza per mezzo di reasoner o entailment ma non ha struttura interna, poichè non ha struttura per le proprietà. Alcuni esempi di triplestore sono *Openlink Virtuoso*, che permette di avere più grafi e di avere quindi strutture quadruple e non triple e *AllegroGraph*;
- **Property Graph** a differenza del triplestore ha una struttura interna a costo però di perdere la possibilità di effettuare inferenza. Alcuni esempi di property graph sono *GraphDB8* e *Neo4j*.

Tra i reasoner più conosciuti vi sono:

- Pellet (OWL DL);
- Racer (OWL DL);
- Hermit;
- Reasoner nativi di Jena, GraphDB8 e AllegroGraph.

Parte II
Esercitazioni.

1 SPARQL.

Si interroga `https://dbpedia.org` per ottenere una lista di vulcani:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT distinct ?v
  FROM <http://dbpedia.org>
 WHERE {
         ?v rdf:type dbo:Volcano .
       }
 LIMIT 100
```

I prefissi si aggiungono inserendo l'url tra simboli `<` e `>`. `rdf:type` può essere sostituito da una `a`:

```
?v a dbo:Volcano .
```

Una query SPARQL è strutturata specificando:

- prefissi
- tipo di query
- (fonte)
- (filtri)
- eventuali altre clausole

Le specifiche SPARQL possono essere ottenute all'indirizzo `https://www.w3.org/TR/rdf-sparql-query/`

Esercizio 1.

Si tenta di trovare il numero di dipendenti Google dalla pagina `http://dbpedia.org/page/Google`:

```
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?num
  FROM <http://dbpedia.org>
 WHERE {
         res:Google dbo:numberOfEmployees ?num .
       }
```

```
# o più semplicemente:
SELECT ?num
  FROM <http://dbpedia.org>
 WHERE {
         dbr:Google dbo:numberOfEmployees ?num .
       }
```

Esercizio 2.

Si cerca di ottenere una lista di trombettisti leader di band:

```
SELECT ?person
 WHERE {
       # soggetto   proprietà       oggetto
         ?person    dbo:instrument  dbr:Trumpet .
         ?person    dbo:occupation  dbr:Bandleader .
       }
```

Esercizio 3.

Si cercano i Paesi con più di due grotte:

```
  SELECT ?country
   WHERE {
           ?cave     rdf:type      dbo:Cave .
           ?cave     dbo:location  ?country .
           ?country  rdf:type      dbo:Country .
         }
GROUP BY ?country
  HAVING (COUNT(?cave) > 2)
```

Esercizio 4.

Si vogliono trovare tutti i software sviluppati da società californiane.

```
SELECT DISTINCT ?software
 WHERE {
         ?company rdf:type dbo:Organisation .
         {           # dbpedia validata
           ?software dbo:developer ?company .
         } UNION {   # dbpedia da validare
           ?software dbp:developer ?company .
         }
         ?software rdf:type dbo:Software .
         ?company dbo:foundationPlace dbr:California .
       }
```

Esercizio 5.

Verificare se Natalie Portman è nata negli Stati Uniti:

```
  ASK
WHERE {
        dbr:Natalie_Portman dbo:birthPlace ?city .
        ?city dbo:Country dbr:United_States .
      }
```

Esercizio 6.

Verificare se Roma è la capitale d'Italia:

```
  ASK
WHERE {
        dbr:Italy dbo:capital dbr:Rome .
      }
```

Esercizio 7.

Verificare se è disponibile la data di nascita di Elizabeth Taylor.

```
  ASK
WHERE {
        dbr:Elizabeth_Taylor dbo:birthDate ?_ .
      }
```

Esercizio 8.

Verificare se Milano è in Germania.

```
  ASK
WHERE {
        dbr:Germany dbo:city dbr:Miland .
      }
```

Esercizio 9.

Verificare se Michelle Obama è sposata con Barack Obama.

```
  ASK
WHERE {
        dbr:Michelle_Obama dbo:spouse dbr:Barack_Obama .
      }
```

Esercizio 10.

Si cercano i giocatori di Football americano che pesano più di 100kg.

```
CONSTRUCT {
            ?x rdf:type dbo:AmericanFootballPlayer .
          }
i    WHERE {
            ?x dbo:weight ?kg .
            FILTER(?kg > 1000000000)
          }
```

2 Protégé.

Progetto *Open Source* per gestire ontologie, permette di caricare direttamente l'ontologia di `dbpedia.org` o di `schema.org`. DBPedia usa più di 400 classi, usandone anche di *sporche* (ovvero ripetute o poco utili). Usando il *reasoner*, sono evidenziate le incongruenze (ovvero classi inconsistenti), che non possono avere dunque istanze.

Costruzione di un'ontologia.

Si costruisce una semplice ontologia per la costruzione di un semplice domino geografico.

`City < Region < Country < Continent`

In Protégé, si definisce una nuova ontologia con `File > New` e inserendo poi sottoclassi di `owl:Thing`. Popolata l'ontologia con delle istanze arbitrarie, si definiscono le proprietà. Si cerca di ridurre al minimo il numero di relazioni (si ha un'intuizione della definizione di contenimento), ma non è possibile tentando di ridurre al minimo le classi. Nel menù `Object properties` è possibile definire nuove proprietà e definire dominio e immagine; si attribuiscono poi le proprietà alle istanze andando su `Entities > Individuals` e cliccando su `Object properties assertions` (a destra).