Da Triple ad Ontologia

Davide Camino

Giugno 2024

Abstract

In questi appunti mostriamo alcuni esperimenti fatti in merito alla traduzione di triple del tipo (*soggetto*, *verbo*, *complemento*) in un'ontologia. Lo scopo di questa traduzione è strutturare i darti in modo formale per poter poi manipolarli attraverso strumenti sofisticati, sia sviluppati ad hoc da noi sia sviluppati da altri come i potenti reasoner che fanno inferenza sulle ontologie.

1 Dati di partenza

Per tradurre i dati grezzi utilizzeremo ℂDuce, un linguaggio di programmazione sviluppato appositamente per manipolare documenti XML. Per questa ragione le triple che andremo a tradurre dovranno essere salvate in formato XML, presentiamo un breve esempio del documento che andremo a tradurre per esemplificarne la struttura:

Listing 1: Dati Grezzi

```xml
<?xml version="1.0"?>
<information>
  <info>
    <subject>Leonardo da Vinci</subject>
    <verb>painted</verb>
    <complement>the Mona Lisa</complement>
    <sex>M</sex>
  </info>
  <info>
    <subject>Jane Austen</subject>
    <verb>wrote</verb>
    <complement>Pride and Prejudice</complement>
    <sex>F</sex>
  </info>
</information>
```

Da questo codice possiamo vedere che i dati di partenza costituiscono semplicemente un set di informazioni o asserzioni senza alcuna struttura.

Notiamo inoltre l'aggiunta di un attributo che specifica il sesso della persona, facciamo questo perché uno dei nostri obbiettivi è quello di valutare i bias di rappresentazione, siamo quindi interessati a verificare se i soggetti di un certo sesso vengono associati a certi verbi più o meno frequentemente dei soggetti del sesso opposto.

2 Ontologia di arrivo

Descriviamo brevemente come sarà fatta l'ontologia che vorremmo ottenere.

2.1 Struttura

L'ontologia che vogliamo creare per dare una struttura formale ai dati grezzi sarà costituita, come ogni ontologia da classi, individui e relazioni.

2.1.1 Classi

L'ontologia che vogliamo ottenere deve rappresentare 2 classi fondamentali:

- Persone: la classe che contiene i soggetti delle nostre triple, rappesenta il concetto di persona e tutte le caratteristiche che derivano dall'appartenere alla specie Homo Sapiens Sapiens;
- Complementi: questa classe rappresenta gli oggetti su cui sono svolte le azioni. è una classe estremamente eterogenea dato che non conterrà solamente oggetti fisici, ma anche astratti quali idee, concetti e pensieri

2.1.2 Individui

Gli individui apparterranno a una delle due classi definite precedentemente, vogliamo fare in modo che se due triple di partenza parlano dello stesso soggetto o dello stesso oggetto questo sia rappesentato da un solo individuo anche nell'ontologia che creiamo. Ad esempio:

- Crick, Franklin e Watson scoprirono la struttura del DNA, da queste 3 affermazioni vorremmo ottenere 3 individui di classe Persona e un solo individuo di classe Complemento;
- Albert Einstein scoprì l'effetto fotoelettrico e ideò la teoria della relativià, da queste 2 triple vorremmo ricavare 2 complementi ma un solo soggetto.

2.1.3 Relazioni

Per ogni tripla vogliamo aggiungere una relazione alla nostra ontologia che colleghi il soggetto al complemento mediante la psecifica azione; Ovviamente vorremmo collegare più soggetti allo stesso complemento (anche mediante azioni differenti) e collegare più complementi allo stesso soggetto.

2.2 Formalismo

Esistono diversi modi per esprimere struttura e contenuto di un'ontologia, alcuni formalismi sono più espressivi di altri e pagano questo aumento di esperessività con una complessità maggiore sia per quanto riguarda la struttura stessa dell'ontologia che per ciò che riguarda gli algoritmi che faranno inferenza e/o reasioning sui dati contenuti nell'ontologia.

Come già detto nella sezione precedente il linguaggio di programmazione che usiamo è specifico per la manipolazione di documenti XML; prendiamo quindi spunto dalle ontologie create con protegé in particolare valuteremo pro e contro di documetni creati attenendosi ai formati di protegé RDF/XML e OWL/XML

3 Linguaggio

Il linguaggio di programmazione scelto per la traduzione è ℂDuce, un linguaggio di programmazione funzionale general porpouse sviluppato appositamente per manipolare documenti XML.

ℂDuce è in grado di fare il parsing della struttura ad albero di un documento XML in modo molto intiutivo, ogni elemento XML avrà la forma $< (tag)(attr) > content$ dove *tag*, *attr* e *content* sono espressioni del linguaggio, in particolare essendo ℂDuce formetmente tipato qualsiasi elemento XML è del tipo `<(Atom) ({..})>[Any*]`.

Riportiamo qui alcune caratteristiche dei tipi citati, perché sono importanti per comprendere le scelte di progettazioni future

- **Atom**: sono elementi simbolici, vengono usati per specificare i nomi dei tag XML e seguono le stesse regole per gli identificatori delle variabili
- **Record (..)**: sono set finiti di (nome, valore) dove i nomi sono label che seguono le stesse convenzioni degli identificatori e i valori sono espressioni
- **Sequences [Any*]**: sono un elemento fondamentale della sintassi di ℂDuce, servono per rappresentare le stringhe di caratteri e il contenuto degli elementi XML. Nonostante la centralità delle liste queste sono solo zucchero sintattico per il tipo Pair)(coppie)

Usiamo questo linguaggio per le garnzie offetre in fase di compilazione in modo da essere sicuri che le funzioni che scriviamo per trasformare elementi ci permenttano di ottenere esattamente il tipo di dato desiderato.

ℂDuce ci permette di descrivere con precisione la stuttura del documento XML di partenza e quella del docimento di arrivo, le funzioni che mapperanno gli input negli ouutput saranno controllate in fase di compilazione e questo ci assicura che la traduzione avvenga coerentemente con i tipi che abbiamo usato per descrivere i documenti.

4 Traduzione

Presentiamo il codice per tradurre le triple di partenza nei due formati presentati. Discutiamo delle implicazioni delle due traduzioni in seguito

4.1 RDF/XML

Listing 2: RDF/XML persona

```
let fun info_to_pers (Info -> Individual)
  <info>[ <subject> sub <verb> v <complement> comp <sex> s ] ->
    let ab = base @ "#" @ sub in
    let class_res = base @ "#Person" in
    let prop_res = base @ "#" @ comp in
    let prop = match v with
      | "painted" -> <ontology:painted rdf:resource=prop_res> []
      | "wrote" -> <ontology:wrote rdf:resource=prop_res> []
      | "discovered" -> <ontology:discovered rdf:resource=prop_res> []
      | "invented" -> <ontology:invented rdf:resource=prop_res> []
      | "married" -> <ontology:married rdf:resource=prop_res> []
      | "built" -> <ontology:built rdf:resource=prop_res> []
      | _ -> <ontology:action rdf:resource=prop_res> []
    in
    <owl:NamedIndividual rdf:about=ab> [
      <rdf:type rdf:resource=class_res> []
      prop
      <rdfs:comment>s
    ]
```

Applicato questa funzione alla prima informazione del Listato 1 oteniamo:

Listing 3: Traduzione persona RDF

```
<owl:NamedIndividual
↪  rdf:about="http://www.semanticweb.org/ontology#Leonardo_da_Vinci">
  <rdf:type rdf:resource="http://www.semanticweb.org/ontology#Person"/>
  <ontology:painted
  ↪  rdf:resource="http://www.semanticweb.org/ontology#the_Mona_Lisa"/>
  <rdfs:comment>M</rdfs:comment>
</owl:NamedIndividual>
```

Con questo codice trasformiamo un elemento di tipo `Info` in un elemento di tipo `Individual`,cioè passiamo da una tripla a un individuo della classe persone. Per completare la traduzione abbiamo poi bisogno di un'altra funzione per trasfomrmare i complementi in induvidui.

Vediamo che in questo formalismo descriviamo l'azione che collega il soggetto al complemento nello stesso elemento XML che descrive anche la persona. Questo permette una sintassi più compatta e leggibile, ma ci costringe a conoscere a prescindere tutti i verbi che possono collegare un soggetto a un complemento (elencati da riga 7 a 13 del Listato 2), tutti i verbi non contemplati possono essere mappati in un solo verbo, perdendo però la specificità dell'azione.

La ragione per cui si devono conoscere i verbi è dovuta al fatto che in questo formalismo il tipo di azione è rappresentato da tag, come abbiamo accennato questi elementi sono di tipo `Atom`, gli elementi di questo tipo rappresentano la struttura ad albero del documento XML e non possono essere creati durante l'esecuzione del programma perché non si potrebbe garantire la correttezza della trasformazione a nel momento della compilazione.

4.2 OWL/XML

Listing 4: OWL/XML informazione

```
let fun info_to_ont (Info -> InfoOnt)
  <info>[ <subject> sub <verb> v <complement> comp <sex> s ] ->
    let subj = "#" @ sub in
    [<Declaration> [<NamedIndividual IRI=("#"@sub)> []]
    <Declaration> [<NamedIndividual IRI=("#"@comp)> []]
    <Declaration> [<ObjectProperty IRI=("#"@v)> []]
    <ClassAssertion> [<Class IRI=("#Person")> [] <NamedIndividual IRI=("#"@sub)> []]
    <ClassAssertion> [<Class IRI=("#Complement")> [] <NamedIndividual IRI=("#"@comp)>
    ↪  []]
    <ObjectPropertyAssertion> [<ObjectProperty IRI=("#"@v)>[]
                               <NamedIndividual IRI=("#"@sub)> []
                               <NamedIndividual IRI=("#"@comp)> []]
    <AnnotationAssertion> [<AnnotationProperty abbreviatedIRI="rdfs:comment"> []
                           <IRI> subj
                           <Literal> s]]

```

Applicato questa funzione alla prima informazione del Listato 1 oteniamo:

Listing 5: Traduzione informazione OWL

```
<Declaration>
  <NamedIndividual IRI="#Leonardo_da_Vinci"/>
</Declaration>
<Declaration>
  <NamedIndividual IRI="#the_Mona_Lisa"/>
</Declaration>
<Declaration>
  <ObjectProperty IRI="#painted"/>
</Declaration>
<ClassAssertion>
  <Class IRI="#Person"/>
  <NamedIndividual IRI="#Leonardo_da_Vinci"/>
</ClassAssertion>
<ClassAssertion>
  <Class IRI="#Complement"/>
  <NamedIndividual IRI="#the_Mona_Lisa"/>
</ClassAssertion>
<ObjectPropertyAssertion>
  <ObjectProperty IRI="#painted"/>
  <NamedIndividual IRI="#Leonardo_da_Vinci"/>
  <NamedIndividual IRI="#the_Mona_Lisa"/>
</ObjectPropertyAssertion>
<AnnotationAssertion>
  <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
  <IRI>#Leonardo_da_Vinci</IRI>
  <Literal>M</Literal>
</AnnotationAssertion>
```

Come si vede in questo caso separiamo la descrizione della persona, da quella dell'azione da quella del complemento, In questo formalismo le azioni non devono essere note a priori, essendo l'azione il valore di un elemento del record e non più un tag.

Come si vede dalla traduzione questo formalismo disperde di più le informazioni e risulta di conseguenza più verboso e meno comprensibile, il vantaggio, come accennato prima è quello di poter descrivere una qualsiasi azione con precisione senza dover conoscere tutte le azioni a priori. Questo

formalismo risulta più espressivo e generale; offre la possibilità di descrivere manipolazioni in modo più flessibile con ℂDuce continuando comunque a poter beneficiare del cotnrollo statico sui tipi offerto dal linguaggio.

4.3 Differenze

Possiamo individuare la differenza principale tra i due formati nella gestione delle relazioni tra soggetto e complemento:

- **RDF**: in questo formato è il soggetto l'elemento principale della nostra descrizione, tra i vari elementi che descrivono il soggetto abbiamo i campi che riguardano le relazioni a cui il soggetto partecipa, in questo senso è il soggetto che è collegato tramite un'azione al complemento. Questo approccio è dovuto al fatto che RDF è un formalismo più di basso livello in cui le relazioni non sono la parte centrale della descrizioni degli individui
- **OWL**: questo formato invece pone enfasi sulle relazioni che si instaurano tra gli individui descritti nell'ontologia; soggetto e complemento sono individui separati e indipendent. L'azione che in questo formato è l'elemento principale della descrizione ed è collegata al soggetto e al complemento in modo da unirli.

5 Manipolazione

Ora che abbiamo ottenuto delle ontologie dalle informazioni partenza possiamo fare in modo da sfruttare la struttura formale per poter fare inferenza sui dati, manipolarli e cercare informazioni in modo più efficace rispetto al set di partenza.

In questa sezione mostriamo come implementare una semplice query per cercare tutti i pittori nella nostra ontologia, sfruttiamo questo esempio molto semplice per confrontare 3 differenti approcci, che sfruttino l'iontologia appena creata.

5.1 Description Logic

Dato che abbiamo ottenuto un'ontologia (a prescindere che sia in formato RDF o WOWL) possiamo adesso sfruttare strumenti molto potensi sviluppati appositamente per lavorare su queste rappresentazioni formali dei concetti. In questo esempio sfruttiamo Protegé e uno dei reasoner offerti da quest'ultimo (HermiT). Grazie al reasoner possiamo interrogare l'ontologia attraverso delle query in Description Logic (DL). Per trovare tutti i pittori ad esempio possiamo semplicemente chidere:
`painted some`
e otterremo come risposta tutti gli individui che partecipano alla relazione "painted"

5.2 ℂDuce & RDF

Vediamo ora come sia possibile ottenere le stesse informazioni utilizzando ℂDuce partendo da un'ontologia in formato RDF attraverso una query.

Le query offerte da ℂDuce sono un potente strumento per cercare informazioni e potenzialmente anche manipolarle. Ricordano molto la sintassi delle query SQL e da queste ereditano anche le tecniche di ottimizzazione basate sulla logica SQL; quando andremo a scrivere una query quindi questa sarà automaticamente ottimizzata. Formalmente tutto ciò che esprimiamo con una query può essere ottenuto con una funzione che si basi sulle `transform`, una funzione di questo genere però non trarrebbe vantaggio dall'ottimizzazione.

Listing 6: Painter RDF

```
let painter = select x
              from x in [ont]/Individual,
                   y in [x]/IndProp
              where (
                match y with
                | <ontology:painted ..>[] -> `true
                | _ -> `false
              )
```

Questo codice ci permette di ottenere la lista delle persone che partecipano alla relazione “painted”; in particolare chiediamo una lista di individui, poi selezioniamo solo gli individui che abbiano una proprietà che faccia il match con l’elemento specificato a linea 6, tutto ciò che non fa il match (catturato dalla wildCard “_”) falsifica la clausola `where`

In questo formato(RDF) siamo costretti a conoscere le azioni che possono collogare soggetti a complementi a priori, questo significa che l’atomo `ontology:painted` deve essere noto e codificato nel programma.

5.3 ℂDuce & OWL

Se il formato dell’ontologia che vogliamo manipolare è OWL possiamo scrivere la funzione di ricerca in questo modo:

Listing 7: Painter OWL

```
1 let painter = select x
2               from x in [ont]/ObjectPropertyAssertion,
3                    y in [x]/ObjectProperty
4               where (y = <ObjectProperty IRI="#painted">[])
```

La struttura della quey è molto simile alla precedente, in questo caso potevamo scegliere di visualizzae solo la dichiarazione dell’individuo pittore, ma abbiamo deciso di visualizzare invece la relazione che collega la persona al complemento, questo rende ancora più snella la query e ci permette comunque di conoscere l’IRI associato al pittore.

5.4 Differenze

Nel secondo caso non è necessrio fare il match, in quanto non stiamo confrontando il tipo di un elemento con quello che stiamo cercando, ma sappiamo esattamente come questo elemento è fatto, quindi è sufficiente la clausola di uguaglianza. Nonostante questo concetto di uguaglianza sembri essere più stringente del concetto di match la seconda query è più flessibile, perché l’elemento che specifica il tipo di azione non è più un atomo, ma un valore che può quindi essere generato “runtime”. La differenza con la query che lavora sul formato RDF è che quella in OWL può essere inclusa in una funzione che accetti come parametro una stringa (stringa che rappresenta l’azione che stiamo cercando) rendendola di fatto molto più flessibile.

Per contro nel formato OWL abbiamo dovuto decidere arbitrariamente cosa la query avrebbe dovuto restituire, infatti mentre nella prima query il risultato ovvio era una lista di individui nella seconda potevamo decidere tra:

- Lista di associazioni (quello che abbiamo effettivamente fatto);
- Lista di individui che però non ci avrebbero dato informazioni sull’oggetto dipinto;
- Combinare le due precedenti eventualmente aggiungendo anche il sesso dell’individuo. Questa sarebbe stata la risposta più completa alla nostra domanda; ma avrebbe complicato notevolmente la query.

Riassumendo: per quanto riguarda le query quindi il formato OWL ci permette di essere più flessibili nella ricerca, ma il fatto che le informazioni siano molto sparse complica notevolmente la costruzione della risposta.

6 Conclusioni

6.1 Risultati ottenuti

In questo lavoro siamo riusciti a arricchire un set di informazioni dando loro una struttura formale grazie alla quale la ricerca e la manipolazione sia supportada da potenti strumenti sviluppati appositamente per le ontologie.

Abbiamo ottenuto differenti formati dell’ontologia, ogniuno con vantaggi e svantaggi specifici. Il primo formato descritto è più rigido per quanto riguarda la creazione e la manipolazione, mentre il secondo ci permette più flessibilità.

Dal punto di vista della complessità i formati si scambiano, il primo è più semplice ed è il più adatto ad altre manipolazioni machine oriented, può ad esempio essere l'input di progrmmi di apprendimento automatico. Il secondo più espressivo risulta più complesso e non particolarmente adatto a algoritmi di AI.

Seppure sia possibile tradurre un formato nell'altro attraverso strumenti differenti da ℂDuce si vorrebbe, per questioni di overhead evitare una catena continua di traduzioni per passare continuamente da un formato all'altro in base a quello più comodo per la specifia elaborazione.

6.2 Lavoro futuro

6.2.1 Duplicati nella traduzione

Per quanto riguarda la traduzione attualmente si fa in modo che due triple differenti rappresentino lo stesso soggetto semplicemente duplicando la rappresentazione, gli strumenti di visualizzazione delle ontologie sono in grado di evidenziare questi duplicati e eliminarli automaticamente, sarà necessario modificare le funzioni di traduzione in modo che questi duplicati non vengano proprio creati.

6.2.2 Struttura dell'ontologia

Dato che adesso stiamo rappresentando i dati con una struttura formale possiamo aggiungere sottoclassi per raggruppare gli individui in base a particolari caratteristiche di nostro interesse, ad esempio per uno studio sui bias di rappresentazione potrebbe essere utile raggruppare le persone in caregorie per valutare se ci sono differenze sostanziali nelle azioni che coinvolgono le varie sottoclassi di persone.