Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Esprimere incertezza, ipotesi, dubbi in RDF con le Congetture

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
ALESSIO ROLFINI

Sessione 3
Anno Accademico 2020/2021

Alla mia famiglia
e ai miei nonni

Introduzione

La semantica di RDF non permette di esprimere punti di vista contraddittori sullo stesso set di dati.

Il problema consiste sostanzialmente nell'impossibilità di esprimere, in RDF, affermazioni il cui valore di verità sia sconosciuto, oppure in contrasto con quello di altre affermazioni, senza però asserirle, poichè questo le renderebbe indubbiamente vere.

Nel corso del tempo, partendo dalla necessità di esprimere statement su altri statement, sono stati prodotti diversi approcci, nessuno dei quali, purtroppo, sembra dare una risposta efficace all'esigenza che potremmo riassumere nel poter *esprimere senza asserire*.

Nella presente dissertazione presenterò una rassegna degli approcci che, costruiti su RDF, provano a fornire metodi per poter esprimere statement su altri statement, analizzandoli anche alla luce della possibilità di esprimere concetti dubbi, privi di uno specifico valore di verità o in contrasto tra loro.

Presenterò ed esaminerò successivamente le caratteristiche delle *Congetture*, una nuova proposta di estensione di RDF 1.1 che permette l'espressione di grafi il cui valore di verità è sconosciuto.

Descriverò infine il modello semantico formale da me specificamente sviluppato per la *Congetture*.

Questa dissertazione è organizzata come segue:

Nel capitolo 1 verrà introdotto il problema dell'espressione in RDF di statement ipotetici o contraddittori; verrà delineato il nostro caso esemplificativo che ricorrerà in tutto l'elaborato e verranno presentati i cardini della

proposta delle Congetture.

Nel capitolo 2 verranno analizzate in dettaglio le tecnologie esistenti ed il loro approccio al problema, evidenziandone le caratteristiche, i punti di forza e di debolezza.

Nel capitolo 3 verrà presentata nel dettaglio la proposta delle Congetture. Ne verranno delineate le basi e i meccanismi a supporto delle operazioni fondamentali. Verranno poi analizzati alcuni scenari più complessi, ovvero le congetture di congetture, le congetture di collassi e i collassi a cascata. Nella sezione 3.4 le Congetture verranno valutate in rapporto alle altre soluzioni, e verranno discussi i criteri di validità della proposta.

Nel capitolo 4 verrà presentato il modello formale semantico delle Congetture. Si partirà dalle basi teoriche e dalla Simple Interpretation di RDF, analizzando ed estendendo il modello per ospitare i costrutti e le regole delle Congetture.

Nel capitolo finale verranno tracciate le conclusioni del lavoro e verrà lanciato uno sguardo a possibili sviluppi futuri.

Indice

Elenco delle figure

Capitolo 1

Esprimere incertezza, ipotesi, dubbi in RDF

RDF è il framework standard delle tecnologie del Web Semantico per la rappresentazione formale delle informazioni e della conoscenza.

I suoi punti di forza sono l'efficienza nella manipolazione e interscambio dei dati e la facilità con la quale i sistemi automatici sono in grado di interpretarlo. È particolarmente indicato per dati strutturati.

Il suo modello di dati ha una sintassi e una semantica. Permette la rappresentazione di grafi diretti, offrendo pertanto compatibilità con algoritmi di navigazione dei grafi.

Oltre a questo, supporta i meccanismi di inferenza.

Essendo modulare, permette l'elaborazione parallela dei dati e può rappresentare informazioni parziali.

RDF può essere usato per elaborare dati provenienti da fonti differenti, fornendo una rappresentazione uniforme di conoscenza distribuita; è anche utile nel dominio dell'automazione delle attività (*task automation*).

RDF supporta una vasta gamma di applicazioni.

Tuttavia, quando è necessario incapsulare metadati all'interno di statement, esprimere statement su altri statement, oppure, come vedremo, nell'espressione di statement senza un chiaro valore di verità, RDF presenta dei

problemi.

In RDF, ogni fatto riguardante un'entità è espresso asserendo una o più triple composte da un soggetto s, un predicato p e un oggetto o.

Quando si asserisce una tripla (s, p, o), si indica che, tra le risorse identificate dal soggetto s e dall'oggetto o, esiste una relazione indicata dal predicato p.

Quando asseriamo una tripla RDF, ciò che è espresso in essa, nel contesto nel quale è espresso, è sempre considerato **vero**.

Questa caratteristica è utilissima quando vogliamo descrivere, ad esempio, il contenuto di un particolare sito web e le sue relazioni, o l'insieme delle pagine che rappresentano un singolo documento logico, o i diritti di proprietà intellettuale delle pagine web, o le preferenze di privacy di un utente, le policy di un sito web, etc...

Nel momento in cui, invece, ci troviamo a voler esprimere statement contraddittori, cominciano le difficoltà. Pur non vietandolo esplicitamente, RDF non offre modalità semplici e dirette per rappresentare statement che non abbiano un valore di verità stabilito e conosciuto.

In generale, però, le interazioni e le conoscenze umane recano sé ipotesi e affermazioni aventi diverse sfumature di incertezza, confitti, disaccordi. Molte di queste ipotesi e affermazioni non raggiungeranno mai uno stato di verità universalmente riconosciuto.

Come caso esemplificativo, useremo un problema molto conosciuto nella letteratura inglese: la paternità delle opere di Shakespeare.

Dobbiamo a Samuel Johnson, uno scrittore, critico e biografo inglese del XVIII secolo, la nozione secondo la quale Amleto è stato scritto nel 1603 da una persona chiamata William Shakespeare.

Questa nozione è stata considerata, per secoli, dalla maggior parte degli studiosi, una solida verità.

All'inizio del XX Secolo, J. Thomas Looney, un insegnante, ha suggerito nel suo libro Shakespeare Identified (1920), che il vero autore della tragedia fosse in effetti Edward De Vere, 17mo Conte di Oxford, e che avesse scritto

l'opera nel 1596.

Questa è nota come "teoria oxfordiana della paternità delle opere di Shakespeare".

Il problema risiede principalmente nelle scarse informazioni inizialmente disponibili sulla vita di William Shakespeare, tant'è che, in uno studio del 1773, lo studioso George Stevens affermava che "Tutto ciò che sappiamo con certezza su Shakespeare è che nacque a Stratford, dove si sposò ed ebbe figli, si recò a Londra dove fece l'attore e scrisse poemi e drammi, tornò a Stratford dove fece testamento, morì e fu seppellito".

Al giorno d'oggi, dopo letteralmente secoli di studio, il consenso del mondo accademico tra gli "Oxfordiani", ovvero coloro che reputano vera la "teoria oxfordiana", e gli "Stratfordiani", che invece sono aderenti alla tesi tradizionale (ovvero che Shakespeare sia realmente esistito), è in favore di questi ultimi.

Tuttavia bisogna dire che nessuna prova definitiva che dimostri o confuti la "teoria oxfordiana" è mai stata prodotta fino ad oggi. Ed è molto probabile che le cose rimangano così indefinitamente.

Ogni tanto l'affascinante "teoria oxfordiana", che comunque offre degli argomenti interessanti [1], riemerge all'attenzione del pubblico. L'ultima volta è accaduto nel 2011 in occasione della distribuzione di un film sull'argomento (Anonymous, di Roland Emmerich).

Come potremmo esprimere in RDF entrambe le affermazioni? Ovvero che Amleto è stato scritto da William Shakespeare **e** che Amleto è stato scritto da Edward De Vere?

Proviamo a far uso della sintassi Trig, tenendo presente che, per brevità, non specificheremo i namespace e considereremo tutti i prefissi come correttamente associati al loro namespace principale, ad esempio *dc* per Dublin Core, *prov* per Prov-O, etc...

```
:Hamlet dc:creator :WilliamShakespeare ;
```

[1] vedere a riguardo
`https://en.wikipedia.org/wiki/Oxfordian_theory_of_Shakespeare_authorship`

```
    dc:created "1603"^^xsd:gYear.

:Hamlet dc:creator :EdwardDeVere ;
    dc:created "1596"^^xsd:gYear .
```

Queste coppie di statement appaiono evidentemente in conflitto.

Possiamo però esprimere almeno due interpretazioni formali consistenti con esse:

- William Shakespeare ha scritto Amleto con Edward De Vere come co-autore;
- William Shakespeare e Edward De Vere erano la stessa persona.

Tuttavia c'è consenso uniforme tra gli studiosi, sia "Oxfordiani" che "Stratfordiani", sul fatto che Edward De Vere e William Shakespeare fossero individui diversi e che non abbiano mai collaborato.

E che quindi al massimo una sola delle due coppie di statement sia vera.

Rappresentare questo tipo di situazioni in RDF è effettivamente complicato.

Il problema è stato affrontato con diverse proposte (Named Graph, Reification, Relazioni n-arie, etc...). Ma nessuna di esse sembra essere sufficientemente semplice. Tutte necessitano invariabilmente di strutture aggiuntive, più o meno complicate, e le soluzioni prodotte difficilmente sono soddisfacenti.

Ad esempio, con i *Named Graph* potremmo separare le due attribuzioni e aggiungere ad esse gli statement di "provenienza". Ma questo non risolverebbe il problema, perché saremmo comunque di fronte a due statement RDF asseriti simultaneamente che affermano "verità" in contrasto tra loro.

Un altro approccio potrebbe essere di sfruttare i *dataset RDF*, ovvero un insieme di *Named Graph* associati ad un grafo di default anonimo (*unnamed default graph*), introdotti dalla specifica RDF 1.1.

Potremmo quindi provare ad esprimere ipotesi, statement senza un preciso valore di verità, oppure contraddizioni, mediante triple all'interno dei diversi

named graph. La specifica dei *dataset* propone diversi approcci per gestire il valore di verità (o la sua mancanza) dei grafi e del grafo di default, ma, come riportato in [Zim14], "[...] non è stata definita una semantica formale per il modello di dati a grafi multipli perché nessuna delle semantiche presentate ha ottenuto un consenso". Non sarebbe chiaro, quindi, il contesto di queste triple all'interno dei grafi, né se il loro contesto sarebbe differente ed indipendente dagli altri *Named Graph* o dal grafo di default.

In definitiva, non c'è un modo diretto di esprimere, all'interno dello stesso dataset, statement che consideriamo veri insieme a statement che riteniamo comunque degni di nota anche se il loro valore di verità non è stabilito.

Il problema di esprimere triple e grafi senza effettivamente asserirli, rimane quindi un problema ancora aperto.

In questa dissertazione è mia intenzione presentare un nuovo approccio sviluppato dal gruppo di lavoro del Prof. Fabio Vitali dell'Università di Bologna, ovvero le *Congetture* (*Conjectures*). Le *Congetture* sono una proposta per una notazione che permette di *esprimere senza asserire* in RDF, basata sui *Named Graphs*, alla quale ho avuto la possibilità di contribuire, in particolar modo sviluppandone la semantica formale che sarà discussa in dettaglio nel capitolo 4.

Oltre alla specifica per esprimere statement senza asserirli, questo approccio include esplicitamente un meccanismo per *affermare* la verità di una congettura, chiamato *collasso alla realtà* (*collapse to reality*).

Una congettura collassata è allo stesso tempo un grafo congetturale **e** un grafo asserito, ed è un modo semplice per gestire situazioni che, espresse inizialmente sotto forma di congetture, devono successivamente essere considerare vere.

L'approccio è costruito attorno a due concetti principali:

- la *Congettura*: un *concetto il cui valore di verità non è disponibile* e la sua rappresentazione;
- il *collasso alla realtà*: un meccanismo per asserire pienamente, in RDF, quando necessario, il valore di verità della Congettura.

Le *Congetture* incapsulano statement RDF o altre congetture in *Named Graph* contrassegnati con specifici identificatori.

Le *Congetture* possono essere scritte in una forma "forte" (*strong form*):

```
CONJECTURE :deVereWroteHamlet {
    :Hamlet dc:creator :EdwardDeVere .
}
```

Oppure, dandoci modo di includerle in dataset RDF 1.1 puri, in una forma "debole" (*weak form*), dove il predicato è espresso sotto forma di un *predicato congetturale*:

```
@prefix conj0001: <http://example.org/exampleDoc#deVereWroteHamlet>

GRAPH :deVereWroteHamlet {
  :Hamlet conj0001:creator :EdwardDeVere .
  conj0001:creator conj:isAConjecturalFormOf dc:creator .
}
:deVereWroteHamlet prov:wasAttributedTo :JThomasLooney .
```

Statement ulteriori che aggiungono informazioni alla Congettura, ma che ne sono esterni, sono detti *grounds* o *ground statements*, che si può tradurre in italiano in *basi*. La tripla `prov:wasAttributedTo` dell'esempio precedente è una di queste *basi*.

Le Congetture collassate sono simili (ma non identiche) alle *annotazioni RDF** [Arn+22], ma, rispetto ad esse, sono più vantaggiose. Ad esempio offrono la possibilità di modellare facilmente situazioni complesse come Congetture di Congetture, Congetture di collassi, e collassi a cascata. Queste situazioni sono piuttosto frequenti nel mondo scientifico, ad esempio nei casi in cui si riportino Congetture altrui formulando altre Congetture su di esse; oppure quando si voglia ipotizzare che determinate Congetture possano essere collassate (quindi diventare vere); oppure quando le Congetture siano

annidate le une nelle altre: in questo scenario il collasso di una Congettura scatenerà ricorsivamente il collasso di quelle menzionate al suo interno, e di quelle menzionate all'interno delle Congetture via via interessate dal collasso, in una sorta di reazione a catena. Tutti questi scenari verranno approfonditi nel capitolo 3.3, e la loro semantica verrà affrontata in dettaglio nella sezione 4.7.

Nel prossimo capitolo analizzeremo lo stato dell'arte delle tecnologie che hanno già affrontato il problema di poter effettuare affermazioni su altri statement. Vedremo che alcune di esse, nello specifico le *Relazioni n-arie*, *N3*, e, in maniera più limitata, *RDF**, si spingono oltre, tentando di rispondere anche all'esigenza di poter esprimere dei concetti senza asserirli. Tuttavia vedremo che questi approcci sono parziali, limitati, oppure eccessivamente complessi.

Capitolo 2

Background scientifico e tecnologico

Fin dalle prime discussioni su RDF 1.0, è emerso il bisogno di esprimere le informazioni riguardo alla *provenienza* (*provenance*) degli statement.

La *provenienza* degli statement, secondo [PRS18], fa riferimento all'intera gamma di informazioni che contribuiscono all'esistenza di un dato, comprendendo tutti gli elementi e le loro relazioni.

Sostanzialmente, dato uno statement, l'esigenza era di essere in grado di esprimere informazioni su di esso. In altre parole, vi era la necessità di poter fare affermazioni su altre affermazioni.

Nel corso del tempo sono stati introdotti diversi approcci a questo problema [SP20; PRS18], basati su RDF puro o proponendone delle estensioni.

Sikos and Philp, nel loro lavoro di rassegna sulla provenienza nelle rappresentazioni della conoscenza [SP20], elencano tutta una serie di sintassi, modelli di dati, sistemi di query e strumenti software per la creazione, manipolazione e interrogazione delle informazioni di provenienza all'interno di dataset e grafi RDF. Gli autori fanno riferimento a 15 modelli di dati e framework di annotazione, analizzando (tra gli altri) i loro modelli formali, l'utilizzo di vocabolari esterni, la granularità delle asserzioni di provenienza, la compatibilità con modelli di query, etc... Inoltre, ragionano su quali

sistemi e quali strumenti possano fornire supporto alle informazioni di provenienza come estensione del modello RDF, così come vocabolari, ontologie e modelli concettuali specifici per il problema.

Orlandi et al. [OGO21] e Hernandez [HHK15] affrontano le differenze tra i vari modelli e valutano i rispettivi benefici in termini di numero di triple generate, spazio di archiviazione, tempo di esecuzione delle query per estrapolare dati di provenienza. Visto che i modelli sono analizzati da un punto di vista pratico (ad esempio, per selezionare un archivio di grafi appropriato), la discussione è limitata ad approcci che permettano di esprimere meta-fatti come triple, inclusi la *Reification*, le *Singleton Properties*, i *Named Graph* e *RDF**

In questo capitolo verrà fatta una rassegna degli approcci esistenti, insieme ai relativi difetti, relativamente alla possibilità di **esprimere senza asserire**. Verrà utilizzato l'esempio visto nel capitolo precedente e, per ogni approccio, ne verranno analizzati gli aspetti rilevanti in termini di semantica, leggibilità e scalabilità.

In dettaglio, verranno discusse le limitazioni della *Reification*, dei *Named Graph*, delle *Relazioni n-arie* (facendo uso dell'ontologia CIDOC-CRM per la costruzione degli statement), delle *Singleton Properties*, di *Notation3* (N3) e di *RDF**.

2.1 Reification

Il primo approccio che analizziamo è la *Reificazione* [Pat14].

Reificazione in RDF, secondo il W3C (`https://www.w3.org/wiki/RdfReification`) significa "scrivere statement RDF riguardanti statement RDF".

Gli statement sono scomposti e riscritti nei loro elementi atomici; gli elementi atomici vengono poi raggruppati insieme e viene dato loro un identificatore unico per l'intero set. L'identificatore può poi essere usato come soggetto o oggetto di altri statement.

Prendiamo il nostro caso esemplificativo, o, per meglio dire, una parte di esso: Lo statement:

```
:Hamlet dc:creator :WilliamShakespeare .
```

e proviamo ad esprimerlo con la Reification aggiungendo l'informazione di provenienza, ovvero che lo statement è stato attribuito a Samuel Johnson.

Potrebbe risultare come segue:

```
_:xxx rdf:type rdf:Statement .
_:xxx rdf:subject :Hamlet .
_:xxx rdf:predicate dc:creator .
_:xxx rdf:object :WilliamShakespeare .
_:xxx prov:wasAttributedTo :SamuelJohnson
```

Nell'esempio un blank node, identificato localmente da `_:xxx`, è un'istanza della classe `rdf:Statement`, ha *Hamlet* come `rdf:subject`, `dc:creator` come `rdf:predicate`, e *William Shakespeare* come `rdf:object`.

Le informazioni aggiuntive di provenienza consistono nello statement di attribuzione a Samuel Johnson (ultima affermazione).

Secondo la specifica RDF, "una *Reification* di una tripla non implica la tripla e non è implicata da essa. La *Reification* dice semplicemente che ciò che contrassegna la tripla esiste, e cosa riguarda, non che sia vero". [Pat14].

Ma questa soluzione sembra essere piuttosto farraginosa, e presenta alcuni seri problemi:

"Prima di tutto, aggiungere quattro triple di reificazione per ogni tripla reificata è inefficiente per lo scambio di dati così come per la gestione di dati RDF che includono metadati a livello di statement. Secondo, scrivere query per accedere ai suddetti metadati è un problema, perché ogni (sub-)espressione relativa ad essi in una query deve essere accompagnata da un'altra sub-espressione per le corrispondenti quattro triple della Reification". [HT14].

La *Reification* non ha semantica formale, e questo porta ad un incremento importante del numero di triple, quindi, non è scalabile.

Infatti la *Reification* richiede almeno 3 statement: uno per il soggetto, un altro per il predicato e un terzo per l'oggetto della tripla. A ciò vanno aggiunti tutti gli ulteriori eventuali statement, ad esempio quelli per la provenienza.

Possiamo facilmente vedere che la quantità di statement nel dataset verrà moltiplicata di almeno un fattore 3, che diventa 4 se includiamo la provenienza.

Questa moltiplicazione di statement è una delle ragioni principali per l'impopolarità della *Reification*, tant'è che sono sorte alcune proposte per la sua deprecazione [SP20].

2.2 Named Graph

La prima raccomandazione del W3C su RDF specifica la nozione di *grafo RDF*, unitamente ai meccanismi per fondere insieme più grafi RDF. Ma non descrive alcun meccanismo per esprimere metadati sui grafi o per esprimere relazioni tra grafi.

In [Car+05] gli autori, cercando di superare i problemi della *Reification*, propongono un'estensione a RDF chiamata *Named Graph*.

Questa estensione consiste in un grafo RDF al quale viene assegnato un nome sotto forma di una *URI Reference*. Una *URI Reference* è una sequenza di caratteri senza caratteri di controllo che consiste in un URI e un frammento identificativo opzionale, ad esempio `http://www.example.org/index.html#section2`, dove la parte che precede il cancelletto è un URI, mentre il resto della stringa è il frammento identificativo.

I *Named Graphs* sono stati inclusi nella specifica RDF 1.1 [LWC14], venendo scelti come uno dei meccanismi (insieme ai dataset RDF) per raggiungere i medesimi obiettivi della *Reification*, che, tra l'altro, **non** è poi stata deprecata. [Haw11]

Un importante vantaggio dei *Named Graph* risiede nel fatto che essi forniscono un *alias* per menzionare insiemi di triple in diversi ambiti e per diversi scopi, tra i quali la provenienza.

La versione 1.1 di SPARQL fornisce supporto per i *Named Graph*, permettendo di effettuare query sul contenuto dei grafi.

Tornando al nostro caso esemplificativo, vediamo come potrebbe essere espresso con i *Named Graph*:

```
GRAPH :attribution01 {
    :Hamlet dc:creator :WilliamShakespeare;
            dc:created "1603"^^xsd:gYear.
}
:attribution01 prov:wasAttributedTo :SamuelJohnson .

GRAPH :attribution02 {
    :Hamlet dc:creator :EdwardDeVere;
            dc:created "1596"^^xsd:gYear .
}
:attribution02 prov:wasAttributedTo :JThomasLooney .
```

Abbiamo due grafi che si chiamano rispettivamente `:attribution01` e `:attribution02`, ognuno dei quali contiene l'attribuzione di Amleto, rispettivamente a William Shakespeare e a Edward De Vere. I grafi sono poi referenziati, tramite il loro nome, nelle rispettive dichiarazioni di provenienza, venendo attribuiti rispettivamente a Samuel Johnson e J. Thomas Looney.

I due grafi permettono di riportare correttamente le informazioni di provenienza. Tuttavia esprimono statement in contrasto tra loro.

I *Named Graph*, rispetto alla *Reification*, forniscono una modalità più facile e naturale per esprimere statement riguardo altri statement. Sfortunatamente, però, non danno alcuna chiara indicazione per discernere la verità (o la falsità) delle loro triple.

2.3 Relazioni n-arie

In RDF e nel Web Semantico in generale, una proprietà è una relazione binaria che collega un elemento ad un altro elemento, o ad un valore. In

RDF, la proprietà p collega un soggetto s ad un oggetto o.

Può accadere che il modo migliore e più naturale per rappresentare determinati concetti sia di collegare un elemento a più elementi e/o a più valori. In questo caso abbiamo una *relazione n-aria*.

Possibili applicazioni delle *relazioni n-arie* possono essere: la descrizione delle qualità (ad esempio la forza) di una relazione binaria; la descrizione di relazioni tra più individui; o la descrizione di una relazione applicata ad una lista di individui.

In [NR06], gli autori sottolineano esplicitamente la differenza tra *Reification* e le *Relazioni n-arie*: la *Reification* si concentra sugli statement, e ne fornisce informazioni aggiuntive.

Le *Relazioni n-arie*, invece, si focalizzano sulla relazione: gli argomenti aggiuntivi non caratterizzano gli statement, piuttosto forniscono ulteriori informazioni riguardo alla relazione stessa.

La rappresentazione di queste informazioni può avvenire seguendo due pattern. Il primo pattern (pattern 1) esprime la relazione sotto forma di una classe, le cui istanze corrispondono alle istanze della relazione. Ulteriori attributi che qualificano la relazione possono essere espressi mediante proprietà aggiuntive che li collegano all'istanza stessa.

Il secondo pattern (pattern 2) esprime invece la relazione associandola ad una sequenza di argomenti. Questo pattern torna utile quando la relazione collega un elemento a più elementi il cui ordine è significativo.

Le *Relazioni n-arie* ci permettono di esprimere statement complessi, come ad esempio la provenienza.

La provenienza può essere considerata come un attributo che descrive una relazione, generando quindi una struttura con più di due endpoint.

In questo caso, la relazione diventa un'istanza di una nuova classe, e gli endpoint diventano proprietà di questa nuova classe. Questo approccio corrisponde al pattern 1 visto poc'anzi.

Come potremmo esprimere il nostro caso esemplificativo con le *Relazioni n-arie*? Può venire in aiuto l'ontologia CIDOC CRM [Bek+21].

L'ontologia CIDOC CRM è un'ontologia formale dedicata al dominio dello scambio di informazioni riguardanti il patrimonio culturale.

E' stata creata per le necessità di documentazione e ricerca dei musei, permettendo lo scambio e l'integrazione di informazioni provenienti da fonti eterogenee come, ad esempio, testi, materiale audio-visivo, e anche dalla tradizione orale.

Seguendo il nostro caso esemplificativo, vediamo come possiamo esprimere con le *relazioni n-arie*:

```
:Hamlet dc:creator :WilliamShakespeare .
```

aggiungendo anche l'informazione di provenienza, ovvero che lo statement viene attribuito a Samuel Johnson.

Abbiamo bisogno di almeno 3 endpoint: l'opera, l'autore e lo statement di attribuzione.

Usiamo quindi la classe CIDOC CRM `E65_Creation` per supportare due o più endpoint come proprietà. Questa classe "comprende eventi che risultano nella creazione di oggetti concettuali o prodotti immateriali, come ad esempio leggende, poesie, testi, musica, immagini, film, leggi, etc...".

L'attribuzione può essere rappresentata come un `E89_Propositional_Object`, che specifica "insiemi di proposizioni su oggetti reali o immaginari che sono documentati come singole unità o che servono come argomento di discorso".

Gli `E89_Propositional_Object` stessi richiedono una attività `E65_Creation` per associarli a chi ha fatto l'affermazione.

Quindi, una rappresentazione della prima parte nostro esempio utilizzando le *Relazioni n-arie* potrebbe essere:

```
:Hamlet a crm:E31_Document .
:Hamlet crm:P94i_was_created_by :CreationOfHamlet.

:CreationOfHamlet a crm:E65_Creation.
:CreationOfHamlet crm:P14_carried_out_by :WilliamShakespeare .
:CreationOfHamlet crm:P4_has_time-span :Year1603 .
```

```
:CreationOfHamlet crm:P67_is_referred_to_by :Attribution01 .

:Attribution01 a crm:E89_Propositional_Object .
:Attribution01 crm:P94i_was_created_by :CreationOfAttribution01.
:CreationOfAttribution01 a crm:E65_Creation.
:CreationOfAttribution01 crm:P14_carried_out_by :SamuelJohnson .
```

Nel primo gruppo di statement vediamo che *Amleto* è un `E31_Document` che è stato creato (`P94i_was_created_by`) da `:CreationOfHamlet`. È interessante notare che la proprietà `P94i_was_created_by` è l'inverso di `P94_has_created`. In [Bek+21] troveremo la proprietà espressa in quest'ultima forma diretta.

Nel secondo gruppo di statement definiamo l'attività della creazione dell'opera "Amleto". Qui affermiamo che è l'attività è stata effettuata (`P14_carried_out_by`) da William Shakespeare nell'anno 1603 e che il *Propositional Object* `:Attribution01` fa riferimento ad essa.

`:Attribution01` è associato alla persona che ha fatto l'affermazione per mezzo di un'ulteriore attività `E65_Creation`.

L'ultimo statement definisce l'attività `E65_Creation` effettuata da Samuel Johnson quando ha fatto l'affermazione.

Il tutto è, naturalmente, espresso in piena forza: CIDOC-CRM non permette l'espressione di attributi congetturali.

In effetti, ci sono proposte che vanno nella direzione di poter esprimere concetti più sfumati. Ad esempio in [NH17], dove, nel dominio dell'archeologia e del patrimonio culturale, gli autori propongo un metodo per assegnare un valore di fiducia alle varie attribuzioni tipo età, materiale, tipo, etc... Questo valore è definito da un numero reale nell'intervallo [0,1] che rappresenta il **livello di fiducia di un ricercatore** nei fatti che sono affermati, e quindi considerati veri. Come anche gli autori sottolineano, tale valore è inerentemente soggettivo. Questo approccio mira a fornire un indicatore di affidabilità piuttosto che un modo per esprimere ipotesi di attribuzioni. Dob-

biamo anche notare che, nonostante la presenza dell'indicatore di affidabilità, gli statement rimangono comunque asseriti con piena forza.

In teoria, le *Relazioni n-arie* permetterebbero l'espressione di statement senza uno specifico valore di verità. Si potrebbero definire nuove classi: tornando all'esempio potremmo generare quelle relative alle congetture dell'attività di creazione dell'opera "Amleto". Il loro valore di verità sarebbe implicito nel significato assegnato alla classe. Ma la prolissità e la complessità delle *relazioni n-arie* fanno sì che questo approccio sia ancora più macchinoso rispetto ai precedenti. Anche le query per interrogare il dataset risulterebbero estremamente complesse.

2.4 Singleton Properties

Abbiamo visto che sia la *Reification* che le *Relazioni n-arie* incrementano considerevolmente il numero di triple RDF, già solo per poter identificare la tripla originale ed esprimere statement aggiuntivi su di essa.

Oltre a questo, nella *Reification*, l'attribuzione originale non appare in alcun modo collegata alla sua controparte reificata. Questo implica che le asserzioni dove l'entità reificata compare come soggetto non sono necessariamente associate all'attribuzione originale.

La necessità di risolvere questi due problemi ha dato luogo ad un approccio completamente diverso, chiamato *Singleton Properties* [NBS14] .

Una *singleton property* è "un'istanza di una proprietà che rappresenta una specifica relazione tra due particolari entità all'interno di uno specifico contesto."

In altre parole, un *singleton* è un predicato unico che è usato solo una volta al fine di rappresentare una tripla al posto del predicato originale.

Visto che è usato per rappresentare una singola tripla, il *singleton* può diventare il soggetto di triple aggiuntive: per costruzione, ogni tripla riguardante il nuovo predicato viene considerata come un'asserzione riguardante l'intera tripla.

Prendiamo nuovamente il nostro caso esemplificativo. Questa volta consideriamo entrambe le attribuzioni, ovvero Amleto creato da William Shakespeare, così come affermato da Samuel Johnson, e Amleto creato da Edward De Vere, così come proposto da J. Thomas Looney.

Invece di `dc:creator`, creiamo due nuovi predicati `xxx:creator#1` e `xxx:creator#2` e otteniamo:

```
:Hamlet  xxx:creator#1  :WilliamShakespeare .
:Hamlet  xxx:creator#2  :EdwardDeVere       .
xxx:creator#1  s:singletonPropertyOf  dc:creator     .
xxx:creator#2  s:singletonPropertyOf  dc:creator     .

xxx:creator#1  prov:wasAttributedTo     :SamuelJohnson .
xxx:creator#2  prov:wasAttributedTo     :JThomasLooney .
```

visto che `s:singletonPropertyOf` è, per definizione, una sottoproprietà di `rdf:type`, entrambi i nuovi predicati `xxx:creator#1` e `xxx:creator#2` sono definiti come proprietà di `dc:creator`.

È interessante notare che, individuato un predicato, possiamo definire un identificatore per l'intera tripla, ad esempio `xxx:creator#1` per
`:Hamlet  xxx:creator#1  :WilliamShakespeare .`

Ciò comporta l'aggiunta di un solo nuovo statement al dataset, ovvero `s:singletonPropertyOf`.

Lo svantaggio principale delle *Singleton Properties* risiede nel fatto che i nuovi predicati sono sottoproprietà delle proprietà originali. Quindi lo statement originale rimane ancora esplicitamente asserito in piena forza. Infatti, nell'esempio, entrambe le attribuzioni rimangono asserite come se avessimo utilizzato il predicato `dc:creator` stesso.

La conclusione è che, anche utilizzando le *Singleton Properties*, non c'è modo di esprimere uno statement senza asserirlo.

2.5 Notation3

Notation3 (o *N3*) è una proposta W3C che consiste in un linguaggio logico per estendere (e, possibilmente, superare) RDF.

Presentata per la prima volta del 2008 ([Ber+08]), dopo un lungo periodo costellato da alterne fortune, N3 è stata nuovamente oggetto di interesse presso la comunità scientifica nel 2021 grazie ad una nuova Nota W3C [Arn+14].

N3 introduce il concetto di *quoted graph*. Un *quoted graph* è un grafo che può essere citato da altre formule. Nella specifica, troviamo affermato esplicitamente che "una formula citata non asserisce come vero il contenuto del grafo RDF (ad esempio `:cervantes dc:wrote :moby_dick`). Infatti, la formula citata è interpretata come una risorsa a sé stante". [Arn+14]

Lo scopo di Notation3 va oltre la semplice specifica di fatti e asserzioni. N3 è anche un linguaggio per **formulazioni logiche** su fatti e asserzioni, e permette di operare dei ragionamenti sulle triple RDF.

La sintassi di N3 si basa fondamentalmente su RDF. È presente il concetto di tripla (s, p, o). Le differenze con RDF si possono trovare nella possibilità di citare i grafi, che diventano "formule" (come nell'esempio che vedremo poco oltre), e nel supporto dei quantificatori matematici .

Le sue caratteristiche possono essere così riassunte:

- N3 è un superset di RDF e Turtle;
- N3 aggiunge uno stile decisionale If-Then sotto forma di implicazioni logiche e variabili;
- N3 supporta le citazioni e le descrizioni di grafi di statement (es: le informazioni di provenienza);
- N3 include un insieme di primitive per l'accesso remoto a informazioni online;

- N3 supporta la *scoped negation-as-failure* [1]

Il nostro caso esemplificativo espresso in N3 diventa:

```
:JThomasLooney suggested { :Hamlet dc:creator :EdwardDeVere; }
```

oppure, adottando uno stile più formale per le informazioni di provenienza:

```
{   :Hamlet dc:creator :EdwardDeVere; }
    prov:wasAttributedTo :JThomasLooney.
```

Diversamente da quanto succede in RDF 1.1, i grafi in N3 non hanno un identificatore, quindi le annotazioni devono essere effettuate sul posto. D'altra parte, i grafi N3 possono annidarsi fino a livelli di profondità arbitrari, mentre in RDF 1.1 i grafi non possono contenere altri grafi.

Esistono proposte (ad esempio [AW19]) per far uso della potente espressività delle regole N3 per rappresentare diversi tipi di semantiche per i *Named Graphs*, facendo affidamento su triple aggiuntive per specificare le interpretazioni attese per ogni tipo di semantica.

Naturalmente, anche se interessante e sofisticato, questo approccio, per poter funzionare, deve poter contare su un'adozione universale degli strumenti e della semantica di N3.

2.6 RDF*

RDF* [Arn+22] è un'iniziativa per estendere RDF con le regole per poter esprimere statement su altri statement mediante una forma più efficiente di *reification*.

[1] negation-as-failure è una regola di inferenza utilizzata per derivare $\neg p$ dall'impossibilità di derivare p. La "scoped negation-as-failure" supportata da N3 è la possibilità, solo per uno specifico documento (o, essenzialmente, per qualche formula astratta) di determinare oggettivamente se esso/essa contenga, o dia la possibilità di derivare, uno specifico fatto.

Entro ampi vincoli, singole triple RDF possono essere riutilizzate all'interno di altre triple. Questo torna particolarmente comodo in caso di statement di provenienza riguardanti singole asserzioni.

In RDF* il nostro caso esemplificativo diventerebbe:

```
<< :Hamlet dc:creator :WilliamShakespeare >>
    prov:wasAttributedTo :SamuelJohnson .

<< :Hamlet dc:creator :EdwardDeVere >>
    prov:wasAttributedTo :JThomasLooney .
```

Sostanzialmente la tripla di attribuzione diventa essa stessa soggetto della tripla di provenienza.

Si vede quindi come la caratteristica fondamentale i RDF* sia la possibilità di utilizzare statement come soggetto di asserzioni distinte.

Una caratteristica di RDF* rilevante ai fini di questa dissertazione è che, da specifica, le due triple `dc:creator` **non sono considerate asserite**.

In RDF*, questi due statement sono definiti come *citati* ma non *affermati*.

Quindi, in effetti, nel nostro esempio, non viene fatta alcuna affermazione riguardo alla paternità dell'opera, e, quindi, le due triple non risultano inconsistenti tra loro.

RDF* mostra in effetti un modo diretto ed estremamente semplice per esprimere affermazioni senza asserirle.

Tuttavia, il limite più importante di RDF* per la nostra problematica è che supporta solo la *citazione* di singole triple.

RDF* è stato sviluppato per fornire a **singole triple** una identità che possa servire come loro riferimento. La regola che permette l'inclusione di singole triple come soggetto è stata concepita specificamente per questo scopo.

I grafi, invece, che sono già forniti di un'identità, rimangono al di fuori del perimetro di RDF*.

In conclusione, l'impossibilità di includere grafi in RDF*, insieme ad una semantica dei grafi poco chiara per quanto riguarda il valore di verità del

contenuto ([Zim14]), comporta il fatto che non ci sia modo (in RDF* così come in RDF) di esprimere **gruppi di statement** senza contemporaneamente asserirli.

2.7 Valutazioni

Abbiamo esaminato i diversi approcci esistenti che permettono di specificare statement su altri statement e, tra di essi, quelli che permettono di esprimere statement senza asserirli.

L'approccio *Reification* scompone singoli statement raggruppandone gli elementi ed assegnando ad essi un identificatore unico. Questo permette di aggiungere ulteriori statement che diano informazioni aggiuntive, ad esempio quelle di provenienza. Questo approccio, però, comporta un notevole aumento del numero di statement nella base di conoscenza. Riguardo all'esigenza di poter esprimere senza asserire, manca completamente il supporto: tutti gli statement risultano sempre asseriti in piena forza.

L'approccio *Named Graph* offre la possibilità di raggruppare più statement all'interno di grafi dotati di un identificatore. Questo permette di esprimere molto facilmente affermazioni su singoli statement e anche su grafi. Il punto di debolezza risiede nel fatto che tutto il contenuto dei *Named Graph* risulta asserito, o meglio, nella specifica non vi è alcun meccanismo esplicito per governare lo stato di verità delle triple contenute nei grafi. I *Named Graph* hanno comunque potenzialità interessanti: vedremo nel prossimo capitolo che l'approccio delle *Congetture* si basa su di essi.

Le *Relazioni n-arie* non si focalizzano sugli statement, ma a livello della relazione tra soggetto e oggetto, quindi alla proprietà, aggiungendo informazioni ad essa. Le proprietà diventano quindi istanze di classi con determinate caratteristiche. Diventa quindi possibile esprimere statement su altri statement sotto forma di attributi alle relazioni di questi statement. Ma, come abbiamo visto, anche con questo approccio siamo di fronte ad una moltiplicazione delle triple RDF necessarie. Con le *Relazioni n-arie* sa-

rebbe anche possibile implementare l'espressione di statement con valore di verità sconosciuto. Questo però comporterebbe la creazione di classi *ad hoc* dedicate, rendendo particolarmente complessa la costruzione e soprattutto l'interrogazione della base di conoscenza

Le *Singleton properties* sono predicati utilizzati una volta sola che rappresentano uno specifico statement, sostituendolo e venendo utilizzati come soggetto in statement successivi, permettendo quindi di specificare asserzioni su di esso. Agiscono a livello di proprietà, venendo definiti come `singletonPropertyOf` dei predicati degli statement originali. A differenza della *Reification*, permettono di mantenere un collegamento esplicito con gli statement originali. Purtroppo, proprio a causa di questo collegamento esplicito, sono inadatti a esprimere statement con valore di verità sconosciuto: le *Singleton property* sono, per definizione, delle **proprietà** dei predicati originali. Le triple espresse con le *Singleton property* rimangono quindi asserite in piena forza esattamente come le triple originali.

La *Notation3* è una proposta che estende RDF introducendo nuovi concetti ed il supporto a formulazioni logiche su fatti e asserzioni. È presente il concetto di grafo, chiamato "formula", che però non è associato ad alcun identiticatore. Questo comporta che l'espressione di statement su altri statement, o meglio, sulle "formule", debba essere effettuata direttamente sul posto. La specifica permetterebbe di esprimere concetti senza uno specifico valore di verità, ma il problema più grosso risiede nel fatto che, divergendo da RDF, tutti gli strumenti di creazione e interrogazione della base di conoscenza dovrebbero essere costruiti specificamente per supportare N3.

*RDF** rappresenta una forma più efficiente di *Reification*, costruita appositamente per esprimere statement su altri statement. Singole triple RDF possono diventare soggetto o oggetto di altre triple RDF, vendendo perciò chiamate *triple citate*. Per quanto riguarda l'espressione di concetti senza uno specifico valore di verità, le *triple citate* sono esplicitamente considerate da specifica **non asserite**. Questo ci permetterebbe di *esprimere senza asserire*, tuttavia *RDF** presenta un'importante carenza: il focus, da specifica, è

su singole triple. Questo rende impossibile esprimere **gruppi di statement** privi di uno specifico valore di verità senza asserirli.

Vedremo nel capitolo successivo una proposta che, basandosi sui *Named Graph*, riesce a dare una risposta semplice e completa al problema di effettuare statement su altri statement senza **necessariamente** asserirli.

Capitolo 3

Le Congetture

La motivazione principale dietro il lavoro che fa da riferimento a questa dissertazione è la possibilità di fare affermazioni su grafi che non necessariamente includono fatti (ovvero asserzioni).

In effetti, mentre nel nostro caso esemplificativo abbiamo ridotto lo scenario ad un singolo statement, ovvero quale tra i personaggi a o b siano stati l'autore di un'opera, gli scenari del mondo reale spesso includono affermazioni più sofisticate che non possono essere ridotte ad una singola relazione binaria.

Non si tratta quindi semplicemente di effettuare affermazioni su grafi, ma di esprimere il fatto che, sebbene non si sia a conoscenza del valore di verità del contenuto di un dato grafo, si vuole comunque essere in grado di esprimere affermazioni su di esso.

Le *Congetture* sono contenitori di triple che rappresentano pienamente la semantica descritta in [Zim14], punto #4: "i named graphs sono considerati 'grafi ipotetici' che recano le stesse conseguenze dei corrispondenti grafi RDF, ma non contribuiscono a definire lo stato di verità del dataset; questo [...] permette al grafo di contenere al suo interno delle contraddizioni senza per questo rendere il dataset contraddittorio".

In altre parole, una congettura è un tipo speciale di *Named Graph* RDF che ospita triple i cui valori di verità non sono definiti.

Analogamente alle *quoted triples* di RDF*, le congetture si rivolgono a triple che sono espresse, ma non asserite.

Tuttavia, le congetture si applicano a grafi RDF piuttosto che solo alle singole triple.

Questo nuovo tipo di *Named Graph* ha lo scopo esplicito di creare grafi ipotetici che **non contribuiscono** al valore di verità dell'intero dataset.

All'interno delle Congetture, tutte le triple devono essere coerenti tra loro, ma il loro valore di verità non è affermato, oppure non è disponibile, ma ciò **non influisce** sul valore di verità complessivo del dataset.

3.1 Grafi Congetturali

L'estensione della sintassi di Trig per supportare le congetture è molto semplice: possiamo sostituire "`GRAPH`" con "`CONJECTURE`" davanti al grafo che vogliamo rendere congetturale. A questo punto nessuna delle triple contenute nel grafo sarà asserita, ma risulterà solo espressa, permettendoci di procedere con ulteriori esplorazioni o ragionamenti.

Di seguito esprimiamo il nostro caso esemplificativo sotto forma di 2 congetture:

```
CONJECTURE :attribution01 {
    :Hamlet dc:creator :WilliamShakespeare;
            dc:created "1603"^^xsd:gYear. }
:attribution01 prov:wasAttributedTo :SamuelJohnson .

CONJECTURE :attribution02 {
    :Hamlet dc:creator :EdwardDeVere;
            dc:created "1596"^^xsd:gYear . }
:attribution02 prov:wasAttributedTo :JThomasLooney .
```

Le due attribuzioni sono espresse come congetture, all'interno delle quali trovano anche spazio ulteriori informazioni (nell'esempio l'anno **ipotetico** di

redazione). Gli unici statement espressi in piena forza sono (giustamente) le triple delle informazioni di provenienza.

Un aspetto chiave è che le congetture **non utilizzano** la *Reification*, né le *Relazioni n-arie*, né classi specifiche create *ad hoc*, quindi sono ortogonali e pienamente compatibili con la maggior parte degli altri approcci.

L'esempio è scritto secondo la sintassi chiamata *strong form*. Esiste però anche una rappresentazione RDF pura delle Congetture, chiamata *weak form*. Questa fa uso di *predicati congetturali*, ovvero predicati appositamente creati "al momento", da utilizzare una sola volta ed esclusivamente nello specifico contesto, che sono associati al predicato originale tramite una mappatura.

La *weak form* corrispondente all'esempio potrebbe essere:

```
GRAPH :attribution01 {
    :Hamlet c01:creator :WilliamShakespeare;
            c01:created "1603"^^xsd:gYear.

    c01:creator conj:isAConjecturalFormOf dc:creator.
    c01:created conj:isAConjecturalFormOf dc:created.
}
:attribution01 prov:wasAttributedTo :SamuelJohnson .

GRAPH :attribution02 {
    :Hamlet c02:creator :EdwardDeVere;
            c02:created "1596"^^xsd:gYear .

    c02:creator conj:isAConjecturalFormOf dc:creator.
    c02:created conj:isAConjecturalFormOf dc:created.
}
:attribution02 prov:wasAttributedTo :JThomasLooney .
```

Nella *weak form*, le congetture sono sia predicati congetturali usati esattamente una volta che una mappatura tra un predicato congetturale e un predicato effettivo mediante la proprietà `conj:isAConjecturalFormOf`.

Al fine di aumentare la leggibilità complessiva, i predicati congetturali possono essere creati in modo che i rispettivi URI abbiano un prefisso differente per ogni grafo (nell'esempio, *c*01 e *c*02), e la medesima parte locale del predicato "effettivo" (*creator* nell'esempio). Questo non è un vincolo tecnicamente obbligatorio, piuttosto una *best practice*.

Si può vedere che i predicati congetturali possono esistere in un ambiente RDF puro con un incremento di triple minimo (due triple RDF per ogni congettura), e i dati possono essere interrogati mediante un motore SPARQL di base, senza particolari estensioni.

I predicati congetturali non sono subordinati ai predicati effettivi. Questo aspetto, oltre ad essere la differenza fondamentale tra le congetture e le *Singleton Properties* [NBS14], permette specificamente la rappresentazione di asserzioni ipotetiche.

L'uso dei predicati congetturali ha due vantaggi: in primo luogo, le affermazioni delle triple originali espresse come congetture *non* vengono asserite, soddisfacendo quindi il requisito fondamentale di poter esprimere senza asserire.

In secondo luogo, vengono asserite affermazioni analoghe, ma diverse, che espongono una relazione (debole) tra gli stessi soggetti e gli stessi oggetti delle affermazioni originali, e che sono in una relazione controllata e ben nota con i predicati originali. Ciò permette una chiara identificazione dei predicati congetturali e una facile interrogazione del dataset.

In tutti gli altri approcci, compresi la *Reification* le *Relazioni n-arie* e RDF*, il dataset della loro forma equivalente (vedere l'esempio per RDF*), non crea una relazione esplicita tra il soggetto e l'oggetto: la query SPARQL più semplice che possa restituire tutte le informazioni disponibili riguardo all'entità `:Hamlet`, non darebbe risultati:

```
SELECT ?p ?o WHERE {
    :Hamlet ?p ?o .
}
```

Evidentemente, `:Hamlet` non è il soggetto di *nessuna* tripla, e, a meno che l'autore della query non voglia esplorare in profondità il dataset, non potrà essere recuperata nessuna informazione (per quanto incerta) riguardo all'entità .

Questa caratteristica può anche essere intenzionale nel disegno dei vari approcci, ma sicuramente renderà più difficile il compito a chi poi va a costruire effettivamente le query di interrogazione del dataset.

Sembra abbastanza evidente, infatti, che una relazione tra `:Hamlet` e `:WilliamShakespeare` esista davvero, così come una relazione tra `:Hamlet` e `:EdwardDeVere`.

Queste relazioni non sono necessariamente attribuzioni di paternità, ma sono comunque relazioni: rappresentano una connessione tra `:Hamlet`, `:EdwardDeVere` e `:WilliamShakespeare`. Questa connessione è presente nel dataset, e dovrebbe poter essere trovata e restituita dalla query enunciata poc'anzi.

Utilizzando le *Congetture*, invece, applicando la query all'esempio all'inizio della sezione, abbiamo come risultato:

```
c01:creator :WilliamShakespeare
c02:creator :EdwardDeVere
```

In effetti il risultato non sembrerebbe proprio chiarissimo, i predicati risultanti, ovvero `c01:creator` e `c02:creator` sono piuttosto oscuri.

Tuttavia, utilizzando una query leggermente più complessa:

```
SELECT ?p1 ?r ?p2 ?o WHERE {
    :Hamlet ?p1 ?o .
    OPTIONAL { ?p1 ?r ?p2 }
}
```

Otteniamo l'informazione completa su `:Hamlet`, ovvero:

```
c01:creator conj:isAConjecturalFormOf dc:creator :WilliamShakespeare
c02:creator conj:isAConjecturalFormOf dc:creator
:EdwardDeVere
```

Cioè: "Amleto è collegato a William Shakespeare attraverso una relazione che è una forma congetturale di "è creato da", ed è connesso a "Edward De Vere" attraverso una *DIVERSA* relazione che è *anch'essa* una forma congetturale di "è creato da". Questi statement sono rilevanti, e sono una rappresentazione corretta di entrambi i dataset, nonchè della realtà.

Abbiamo affermato precedentemente che le forme "strong" e "weak" sono completamente equivalenti.

La conversione di un grafo congetturale in *strong form* in un grafo puro RDF viene effettuata sistematicamente applicando delle regole di conversione.

Regola di conversione delle Congetture #1: ogni tripla `(s, p, o)` all'interno di un grafo congetturale genera un nuovo predicato congetturale *cp*, non utilizzato in nessun'altra parte, ed è convertita esattamente in due triple: `(s, cp, o)` e `(cp, conj:isAConjecturalFormOf, p)`.

3.2 Collasso di congetture

Può succedere che, talvolta, gli studiosi giungano finalmente a delle conclusioni e affermino la verità di una congettura.

Facendo sempre riferimento al nostro esempio, nel caso della diatriba riguardante la teoria Oxfordiana, nel corso del tempo la maggior parte della comunità scientifica ha trovato un consenso generale nell'affermare che Amleto sia stato scritto quasi certamente da William Shakespeare. E che quindi Samuel Johnson aveva ragione.

Per gestire questa situazione, ovvero, per esprimere il fatto che una congettura sia accettata, mentre le altre congetture in competizione vengano rifiutate, si utilizza uno speciale meccanismo chiamato "collasso alla realtà".

Questo termine è mututato dal concetto della meccanica quantistica del "collasso della funzione d'onda di una particella", che si verifica quando, per

effetto di un'osservazione dal mondo esterno, gli autostati della particella osservata si riducono, "collassano", ad uno solo.

Nelle congetture, il concetto significa ridurre le diverse congetture indipendenti e sovrapposte ad una sola di esse, che è considerata quella vera.

Nella "Strong Form", si utilizza la keyword `COLLAPSED` anteponendola a `CONJECTURE` come mostrato di seguito:

```
COLLAPSED CONJECTURE :attribution01 {
    :Hamlet dc:creator :WilliamShakespeare;
            dc:created "1603"^^xsd:gYear. }
:attribution01 prov:wasAttributedTo :SamuelJohnson .
```

Nella *weak form*, ovvero la forma esprimibile in RDF puro, un grafo congetturale collassato è composto da due grafi, rispettivamente:

- la congettura originale, impostata seguendo la regola di conversione #1 vista precedentemente;
- il collasso effettivo, contenente le triple congetturali espresse in piena forza, insieme ad una tripla `conj:collapses` che collega il grafo collassato al grafo congetturale. Per semplicità è chiarezza, la tripla viene posizionata all'interno del grafo collassato, ma non è obbligatorio. può anche essere posizionata altrove.

Vediamo un esempio pratico.
Partendo da questi grafi:

```
GRAPH :attribution01 {
    :Hamlet c01:creator :WilliamShakespeare;
            c01:created "1603"^^xsd:gYear.

    c01:creator conj:isAConjecturalFormOf dc:creator.
    c01:created conj:isAConjecturalFormOf dc:created.
}
```

```
:attribution01 prov:wasAttributedTo :SamuelJohnson .

GRAPH :attribution02 {
    :Hamlet c02:creator :EdwardDeVere;
            c02:created "1596"^^xsd:gYear .

    c02:creator conj:isAConjecturalFormOf dc:creator.
    c02:created conj:isAConjecturalFormOf dc:created.
}
:attribution02 prov:wasAttributedTo :JThomasLooney .
```

Collassiamo il grafo `:attribution01`, dichiarandolo vero. Aggiungiamo quindi al dataset:

```
GRAPH :collapseOfAttribution01 {
    :Hamlet dc:creator :WilliamShakespeare .
            dc:created ‘‘1603"^^xsd:gYear.
    :collapseOfAttribution01 conj:collapses :attribution01 .}
```

Possiamo quindi enunciare La regola di conversione per le *Congetture Collassate*:

Regola di conversione delle Congetture #2: un grafo `COLLAPSED CONJECTURE` $c1$ viene convertito in due grafi: il primo è la forma convertita del grafo $c1$ seguendo la regola #1, e il secondo è un nuovo grafo $cc1$ contenente tutte le triple di $c1$ con i rispettivi predicati originali effettivi, tranne `conj:isAConjecturalFormOf`, più la tripla `(cc1, conj:collapses, c1)`.

3.2.1 Possibili alternative

Costrutti simili alle congetture collassate sono le *annotazioni RDF** [Arn+22]. Riprendendo ancora una volta l'esempio:

```
GRAPH :attribution01 {
    :Hamlet c01:creator :WilliamShakespeare;
            c01:created "1603"^^xsd:gYear.

    c01:creator conj:isAConjecturalFormOf dc:creator.
    c01:created conj:isAConjecturalFormOf dc:created.
}
:attribution01 prov:wasAttributedTo :SamuelJohnson .

GRAPH :attribution02 {
    :Hamlet c02:creator :EdwardDeVere;
            c02:created "1596"^^xsd:gYear .

    c02:creator conj:isAConjecturalFormOf dc:creator.
    c02:created conj:isAConjecturalFormOf dc:created.
}
:attribution02 prov:wasAttributedTo :JThomasLooney .
```

Con le *annotazioni RDF**, risulterebbe espresso in un modo simile (ma non equivalente) alle *congetture collassate*:

```
:Hamlet dc:creator :WilliamShakespeare
    {| prov:wasAttributedTo :SamuelJohnson |} .
:Hamlet dc:created "1603"^^xsd:gYear
    {| prov:wasAttributedTo :SamuelJohnson |} .

<< :Hamlet dc:creator :EdwardDeVere >>
    prov:wasAttributedTo :JThomasLooney .
<< :Hamlet dc:created :"1596"^^xsd:gYear >>
    prov:wasAttributedTo :JThomasLooney .
```

Che, per costruzione, è equivalente a:

```
<< :Hamlet dc:creator :WilliamShakespeare >>
    prov:wasAttributedTo :SamuelJohnson .
:Hamlet dc:creator :WilliamShakespeare.
<< :Hamlet dc:created "1603"^^xsd:gYear >>
    prov:wasAttributedTo :SamuelJohnson .
:Hamlet dc:created "1603"^^xsd:gYear.

<< :Hamlet dc:creator :EdwardDeVere >>
    prov:wasAttributedTo :JThomasLooney .
<< :Hamlet dc:created :"1596"^^xsd:gYear >>
    prov:wasAttributedTo :JThomasLooney .
```

La rappresentazione equivalente delle triple annotate è specificata due volte: una volta tra virgolette (ovvero *congetturata*), come soggetto della tripla `prov:wasAttributedTo`, e un'altra volta asserita come tripla RDF a sè stante.

Differentemente dalle Congetture, le due triple sono completamente indipendenti l'una dall'altra, e non sono collegate tra loro in nessun modo.

3.3 Utilizzi avanzati delle Congetture

Riassumendo, i grafi congetturali `CONJECTURE { ... }` e i grafi collassati `COLLAPSED CONJECTURE { ... }` sono nuovi costrutti introdotti al fine di migliorare l'espressività.

Nella *weak form*, le Congetture richiedono due nuovi predicati, `conj:isAConjecturalFormOf` e `conj:collapses`, e una sorgente di nuovi predicati creati specificamente per essere utilizzati all'interno dei grafi congetturali.

Vediamo adesso una serie di utilizzi avanzati delle Congetture, che integrano e completano le possibilità offerte da questo paradigma.

Analizzeremo la possibilità di effettuare congetture su altre congetture, di effettuare congetture di collassi e collassi a cascata, definendo le caratteri-

stiche di ognuno di questi utilizzi. Valuteremo anche l'esistenza di eventuali alternative in altri paradigmi, presentandole brevemente.

3.3.1 Congetture di Congetture

Negli ambienti scientifici, gli studiosi non esprimono necessariamente congetture solo su affermazioni proprie. Potrebbe capitare che riportino le congetture di qualcun altro, magari trovandosi in disaccordo, ed esprimendo ricorsivamente congetture riguardo ad esse.

Ad esempio, consideriamo lo statement:

> Io **prendo atto** che l'articolo <https://bit.ly/3wSH76A>**riporta** che J. Thomas Looney **ha proposto** che Amleto è stato scritto da Edward De Vere.

A questo scenario si applicano tre livelli di attribuzione: l'ipotesi di paternità di Amleto, la sua proposta da parte di J. Thomas Looney e l'articolo che riporta il tutto.

Vincolare le informazioni di provenienza ad uno solo dei tre livelli è sicuramente limitante.

Le Congetture supportano catene di riferimenti annidati, permettendo di posizionare le triple di provenienza di un determinato grafo all'interno di altre congetture nella catena, come segue:

```
CONJECTURE :attribution02 {
   :Hamlet dc:creator :EdwardDeVere .
}
CONJECTURE :attribution03 {
   :attribution02 prov:wasAttributedTo :JThomasLooney .
}
CONJECTURE :attribution04 {
   :attribution03 prov:wasInformedBy <https://bit.ly/3wSH76A> .
}
:attribution04 prov:wasAttributedTo :me .
```

Vediamo infatti che `:attribution03` specifica la provenienza di `:attribution02` e `:attribution04` specifica la provenienza di `:attribution03`.

Possibili alternative

Anche RDF* permette sequenze (o citazioni annidate)[Arn+22], come, ad esempio:

```
<<
    <<
        << :Hamlet dc:creator :EdwardDeVere >>
            prov:wasAttributedTo :JThomasLooney
    >> prov:wasInformedBy <https://bit.ly/3wSH76A>
>> prov:wasAttributedTo :me .
```

Anche se la sintassi è sicuramente più convoluta.

3.3.2 Congetture di collassi

In alcuni casi, gli studiosi potrebbero credere che una determinata congettura possa essere vera, ma non ne siano completamente certi.

Potrebbero quindi preferire di esprimere il fatto che il loro giudizio semplicemente *propenda* verso tale verità.

Questo è uno schema dialettico piuttosto frequente nei discorsi umanistici: nel dibattito su argomenti ancora aperti, gli studiosi elencano una serie di affermazioni concorrenti come ipotetiche, specificandone la provenienza, per poi alla fine affermare la preferenza per una di esse.

Consideriamo la seguente affermazione:

> L'Enciclopedia Britannica (`https://bit.ly/3qgakFT`) **sostiene** che l'**attribuzione** di Amleto a William Shakespeare da parte di Samuel Johnson sia corretta.

Nella nostra terminologia, questo corrisponde a fare una congettura di un collasso: l'Enciclopedia Britannica fa una congettura del collasso della congettura riguardante l'attribuzione fatta da Samuel Johnson.

Le Congetture supportano pienamente questo schema: visto che i collassi possono essere espressi nella *weak form* come *Named Graph* contenenti il predicato `conj:collapses`, possono essere essi stessi soggetti a congettura, ad esempio:

```
CONJECTURE :attribution01 {
    :Hamlet dc:creator :WilliamShakespeare .
}
CONJECTURE :collapseOfAttribution01 {
    :attribution01 prov:wasAttributedTo :SamuelJohnson .
    :collapseOfAttribution01 conj:collapses :attribution01.
}
:collapseOfattribution01 prov:wasInformedBy
    <https://bit.ly/3qgakFT> .
```

Possibili alternative

Non sembra esistere una rappresentazione equivalente in nessuno degli altri approcci visti precedentemente.

Per esempio, in RDF*, questo schema non sembra essere possibile, visto che richiederebbe la citazione di una annotazione, cosa non permessa.

Nella sintassi specificata nella sezione 3.2.1 della specifica draft di RDF* [Arn+22], la produzione `annotation` non può essere posizionata all'interno di una `quotedTriple`.

Inoltre, in RDF*, le annotazioni sono un modo abbreviato per esprimere due triple separate, mentre le citazioni RDF* funzionano solo su triple singole.

3.3.3 Collassi a cascata

Una volta che permettiamo livelli multipli di annidamento delle Congetture, il collasso di un grafo congetturale diventa un'operazione potenzialmente impegnativa.

Il collasso di una *Congettura di una Congettura* richiede di collassare ricorsivamente tutte le Congetture di Congetture, e di continuare a cascata su tutte le Congetture che troviamo menzionate all'interno delle Congetture via via interessate dalle operazioni.

Questa complessità, tutto sommato, è la rappresentazione diretta della natura delle relazioni di fiducia: se Alice crede nella congettura di Bruno con_B, che a sua volta supporta la congettura di Carlotta con_C, è evidente che anche Alice si fidi della congettura di Carlotta riguardante con_C, e così via, a cascata, per tutte le congetture credute vere da Carlotta.

Le Congetture possono gestire una situazione del genere collassando la congettura del collasso e, ricorsivamente, collassando tutte le triple menzionate dalle congetture - inclusi gli altri collassi.

Ad esempio, riprendendo il listato della sezione precedente, dove riportiamo che, secondo l'Enciclopedia Britannica, l'attribuzione di Amleto fatta da Samuel Johnson **potrebbe** essere corretta:

```
CONJECTURE :attribution01 {
    :Hamlet dc:creator :WilliamShakespeare .
}
CONJECTURE :collapseOfAttribution01 {
    :attribution01 prov:wasAttributedTo :SamuelJohnson .
    :collapseOfAttribution01 conj:collapses :attribution01.
}
:collapseOfattribution01 prov:wasInformedBy
    <https://bit.ly/3qgakFT> .
```

Diciamo che ci fidiamo dell'Enciclopedia britannica, quindi collassiamo la seconda congettura. Siamo sempre in *strong form*:

```
CONJECTURE :attribution01 {
    :Hamlet dc:creator :WilliamShakespeare .
}
COLLAPSED CONJECTURE :collapseOfAttribution01 {
    :attribution01 prov:wasAttributedTo :SamuelJohnson .
    :collapseOfAttribution01 conj:collapses :attribution01 .
}
:collapseOfattribution01 prov:wasInformedBy
    <https://bit.ly/3qgakFT> .
```

Al di là del formalismo sintattico, ci troviamo davanti ad una maggiore complessità, visto che il collasso ha effetto anche su tutti i predicati `conj:collapses` incorporati. Dobbiamo quindi procedere a collassare a cascata tutte le congetture coinvolte. In questo caso specifico, questo significa che, se ci fidiamo dell'Enciclopedia Britannica, allora dobbiamo credere che l'attribuzione di Samuel Johnson sia corretta, e quindi anche che William Shakespeare sia effettivamente l'autore di Amleto.

Il collasso precedente può essere rappresentato in *weak form* come segue:

```
GRAPH :attribution01 {
  :Hamlet c03:creator :WilliamShakespeare .
  c03:creator conj:isAConjecturalFormOf     dc:creator .
}
GRAPH :collapseOfAttribution01 {
  :attribution01 c04:wasAttributedTo :SamuelJohnson .
  :collapseOfAttribution01 c04:collapses :attribution01 .

  c04:wasAttributedTo conj:isAConjecturalFormOf
        prov:wasAttributedTo .
  c04:collapses conj:isAConjecturalFormOf conj:collapses  .
}
:collapseOfAttribution01 prov:wasInformedBy
```

```
          <https://bit.ly/3qgakFT> .

  GRAPH :collapseOfCollapseOfAttribution01 {
    :attribution01 prov:wasAttributedTo :SamuelJohnson .

    :collapseOfAttribution01 conj:collapses :attribution01  .

    :collapseOfCollapseOfAttribution01  conj:collapses
             :collapseOfAttribution01 .

    # cascaded triple
    :Hamlet dc:creator :WilliamShakespeare .
  }
```

Per passare alla *weak form* si userà la seguente regola di conversione.

Regola di conversione delle Congetture #3: una tripla `conj:collapses` all'interno di un grafo `COLLAPSED CONJECTURE` aggiunge ricorsivamente al grafo di collasso la forma "effettiva" (non congetturale) di tutte le triple del grafo `CONJECTURE` che sono oggetto delle triple `collapses`, con l'esclusione delle triple `conj:isAconjecturelFormOf`.

I collassi a cascata sono un meccanismo per rivalutare tutti gli statement coinvolti, direttamente o indirettamente, in un collasso di una congettura.

L'ultimo esempio può essere spiegato in questo modo:

- i due grafi nella *strong form* diventano tre grafi in RDF puro;
- il primo `GRAPH` è la *weak form* di `:attribution01` dopo l'applicazione della regola di conversione #1 alla tripla `dc:creator`;
- il secondo `GRAPH` è `:collapseOfAttribution01` espresso in *weak form* dopo l'applicazione della regola di conversione #1 ai predicati `prov:wasAttributedTo` e `conj:collapses`;

- la prima parte del terzo `GRAPH` è la *weak form* del collasso di `:collapseOfAttribution01` dopo l'applicazione della regola di conversione #2: le triple `prov:wasAttributedTo` e `conj:collapses` sono riformulate nella loro forma effettiva, e una nuova tripla `conj:collapses` viene aggiunta per collegare la congettura `:collapseOfAttribution01` al grafo di collasso `:collapseOfCollapseOfAttribution01`;
- per finire, visto che in `:collapseOfAttribution01` uno dei predicati è appunto `conj:collapses`, viene applicata la regola di conversione #3, il collasso collassa a cascata nella `:attribution01`, e la tripla `dc:creator` viene riformulata nella sua forma effettiva.

3.4 Valutazioni, comparazioni, criteri di validità

Il problema fondamentale affrontato dalle Congetture è la possibilità di esprimere statement RDF senza asserirli. I *Named Graph* sono stati proposti e aggiunti a RDF proprio per questo motivo. Sfortunatamente, i *Named Graph* sono stati interpretati e implementati in modi diversi nel corso del tempo, tanto che al momento è difficile separare le logiche dell'espressione senza asserzione dal corpo della specifica dei *Named Graph* e identificare una logica univoca per il loro uso.

Riprendendo un esempio visto nella sezione 2.2:

```
GRAPH :attribution01 {
    :Hamlet dc:creator :WilliamShakespeare;
            dc:created "1603"^^xsd:gYear.
}
:attribution01 prov:wasAttributedTo :SamuelJohnson .

GRAPH :attribution02 {
    :Hamlet dc:creator :EdwardDeVere;
```

```
            dc:created "1596"^^xsd:gYear .
}
:attribution02 prov:wasAttributedTo :JThomasLooney .
```

La specifica *Named Graph* ci permette di porre i due statement `dc:creator` all'interno dei due grafi distinti, piuttosto che al di fuori di essi, nel grafo di default. Ciò ci permette di associare a loro le informazioni di provenienza, di raggrupparli dando loro nomi distinti, in modo che altre annotazioni possano fare riferimento alle triple in essi contenuti.

Quello che la specifica dei *Named Graph* non fornisce, invece, è la possibilità di metterli in grafi distinti **per il fatto di avere valori di verità diversi**.

La nota W3C [Zim14] identifica otto diverse semantiche che possono essere associate ai dataset:

1. *Named Graphs non aventi significato (3.1)*: in questa semantica il valore di verità dei Named Graph non è considerato ai fini del valore di verità complessivo, che dipende soltanto dal valore di verità del grafo di default;

2. *grafo di default come unione o fusione (3.2)*: in questa semantica il valore di verità complessivo dipende dal valore di verità del grafo di default unito a quello di tutti i named graph;

3. *il nome del grafo denota il named graph o il grafo (3.3)*: in questa semantica il nome dei grafi identifica i singoli grafi, ed è possibile referenziarlo all'interno di altri grafi per fornire, ad esempio, metadati;

4. *ogni Named Graph definisce il proprio contesto (3.4)*: in questa semantica il valore di verità delle triple di ogni grafo è limitato all'interno del proprio *Named Graph*, permettendo di ragionare su concetti non certi o non affidabili senza che questo influisca sul valore di verità complessivo;

5. *i Named Graph sono in una particolare relazione con ciò a cui il nome del grafo fa riferimento (3.5)*: in questa semantica lo stato di verità

del dataset dipende dallo stato di verità della risorsa alla quale fa riferimento; un esempio potrebbe essere una risorsa nel Web: in questa semantica lo stato di verità può cambiare nel tempo;

6. *Quad semantics (3.6)*: questa semantica estende quella di RDF considerando i Named Graph come quadruple composte da soggetto, oggetto, predicato, IRI. La relazione sottesa dal predicato non è più binaria, ma diventa ternaria;

7. *Quoted graphs (3.7)*: questa semantica offre un modo di associare informazioni ad uno specifico grafo RDF "citandolo" utilizzando un letterale che rappresenta sintatticamente il grafo;

8. *relazioni con l'entailment regime di SPARQL (3.8)*: questa semantica si basa su una proprietà delle query SPARQL ASK senza variabili quando, nella clausola WHERE, contengono solo grafi; se la query Q eseguita su un grafo G restituisce *true* all'interno di un regime E, allora si può affermare che $G\ E - entails\ Q$, o il contrario se *false*.

La semantica che meglio affronta il problema qui discusso è *3.4 - ogni named graph definisce il proprio contesto*: "I Named Graph nei dataset RDF sono talvolta usati per delimitare un contesto nel quale le loro triple sono vere. [...] Un esempio di questa situazione si ha quando si vuole tener traccia dell'evoluzione nel tempo di alcuni fatti. Un altro esempio è quando si vuole poter esprimere diversi punti di vista per ragionarci sopra, senza creare conflitti o inconsistenze". Tuttavia, allo stesso tempo, ci sono molti utilizzi dei *Named Graph* nei quali il contenuto dei grafi debba invariabilmente essere considerato vero e definito.

La proposta principale presentata in questa dissertazione è un modo differente di esprimere dei *Named Graph* secondo il punto 3.4.

Le Congetture rappresentano *Named Graph* contenenti triple che, in modo chiaro e trasparente, non sono asserite globalmente, ma sono vere solo all'interno del loro grafo. La proposta non modifica la semantica dei named

graph né aggiunge nuovi concetti a RDF, ma lavora all'interno del modello standard RDF 1.1.

Anche le implicazioni legate alla semantica 3.8 sono molto utili nel nostro caso. Se un motore SPARQL riconosce il grafo come vero utilizzando una query ASK, il grafo deve essere considerato vero. Altrimenti falso. Questo può essere utilizzato come test pratico per verificare se il modello proposto si posiziona bene all'interno dell'ambiente RDF e dei suoi strumenti.

Per provare tramite la semantica 3.8 che le Congetture implementino con successo la semantica 3.4 verifichiamo, attraverso il coinvolgimento dello schema di implicazioni di SPARQL, [Haw+13] che:
a) *i grafi congetturali non siano considerati veri*, e
b) *i grafi congetturali collassati siano considerati veri*;
in più è chiaro che i grafi non congetturali manterranno la medesima (complessa) semantica di prima.

Abbiamo affermato in precedenza che per la *strong form*, la quale fa uso delle keyword aggiuntive `CONJECTURE` e `COLLAPSED CONJECTURE`, esiste sempre una equivalente *weak form* che usa i concetti standard RDF 1.1. Effettueremo quindi i test basandoci sulla *weak form*.

Secondo la semantica 3.8, possiamo testare il dataset effettuando delle query di tipo ASK sui grafi. Quindi, per verificare quale tra Edward De Vere o William Shakespeare sia l'autore di Amleto dichiarato come effettivo, prendiamo l'esempio in *weak form* già visto precedentemente (3.2), al quale aggiungiamo il collasso:

```
GRAPH :attribution01 {
    :Hamlet c01:creator :WilliamShakespeare;
            c01:created "1603"^^xsd:gYear.

    c01:creator conj:isAConjecturalFormOf dc:creator.
    c01:created conj:isAConjecturalFormOf dc:created.
}
:attribution01 prov:wasAttributedTo :SamuelJohnson .
```

```
GRAPH :attribution02 {
    :Hamlet c02:creator :EdwardDeVere;
            c02:created "1596"^^xsd:gYear .

    c02:creator conj:isAConjecturalFormOf dc:creator.
    c02:created conj:isAConjecturalFormOf dc:created.
}
:attribution02 prov:wasAttributedTo :JThomasLooney .

GRAPH :collapseOfAttribution01 {
    :Hamlet dc:creator :WilliamShakespeare .
            dc:created "1603"^^xsd:gYear.
    :collapseOfAttribution01 conj:collapses :attribution01 .}
```

ed eseguiamo la query:

```
ASK WHERE {
    GRAPH ?g1 { :Hamlet dc:creator :EdwardDeVere. }
}
```

La query ritonerà `false`, ovvero il grafo congetturale `:attribution02` non è considerato asserito.

Al contrario la query:

```
ASK WHERE {
    GRAPH ?g2 { :Hamlet dc:creator :WilliamShakespeare. }
}
```

ritornerà `true`, ovvero, il grafo collassato `:collapseOfAttribution01` è considerato asserito.

Questo prova che le Congetture, così come sono proposte, forniscono un'espressione semplice e affidabile dei *Named Graph*, seguendo la semantica 3.4

e, allo stesso tempo, rimanendo compatibili con lo schema di implicazioni di SPARQL così come richiesto dalla semantica 3.8

Questo tipo di validazione è necessario, ma non sufficiente. Abbiamo bisogno di verificare che la nostra proposta sia compatibile anche con il data model di RDF [LWC14].

Secondo [Pat14], un'estensione semantica di RDF corretta consiste in un insieme di assunti semantici che diano agli IRI significati aggiuntivi sulla base di altre specifiche o convenzioni. Quando questo succede, le estensioni semantiche devono conformarsi alle condizioni minime di verità già enunciate, estendendole per accogliere i significati aggiuntivi.

Nel prossimo capitolo vedremo la definizione di una semantica per le Congetture, che rappresenta il mio personale contributo originale alla proposta presentata in questa dissertazione. L'approccio seguito è composto da due parti: prima di tutto verrà estesa l'interpretazione semplice di RDF 1.1 mediante l'inserimento di nuovi e distinti insiemi di IRI per le *proprietà congetturali*, di un insieme distinto di regole per gestirle, e di una specifica mappatura. Questa mappatura modellerà la relazione tra le *proprietà congetturali* e le *proprietà effettive* delle quali le prime sono forma congetturale, sia nella loro forma non collassata che quando sono asserite in piena forza. In secondo luogo verrà analizzata la semantica delle triple congetturali e dei grafi quando vengono effettuate le operazioni di congettura e di collasso, generalizzando e definendo le condizioni per la loro validità.

Capitolo 4

Semantica delle Congetture

La definizione di un modello semantico formale per le Congetture è un requisito fondamentale al fine di verificare la correttezza della proposta rispetto a RDF 1.1.

Partiremo dal modello formale teorico-insiemistico, nel quale verranno delineate le fondamenta della proposta, le definizioni e le proprietà di base.

La *Simple Interpretation* di RDF 1.1 verrà analizzata ed estesa aggiungendo gli insiemi, le funzioni e le condizioni semantiche **di base** delle Congetture.

La nuova *Interpretation* verrà poi verificata applicandola a casi ed esempi.

Nelle sezioni successive l'*Interpretation* verrà ulteriormente estesa e verificata aggiungendo il supporto alle caratteristiche aggiuntive (grafi congetturali, *blank nodes*) e alle operazioni (collasso alla realtà, e le applicazioni avanzate).

4.1 Modello formale teorico-insiemistico

Dati gli insiemi disgiunti $\mathcal{I}$ (tutti gli IRI), $\mathcal{B}$ (blank nodes), and $\mathcal{L}$ (letterali). Una tripla RDF è una tupla $(s, p, o) \in \mathcal{T} = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$.

Per ogni predicato RDF p, sia $\mathcal{S}_p \subseteq (\mathcal{I} \cup \mathcal{B})$ il suo dominio e $\mathcal{O}_p \subseteq (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ la sua immagine.

Denotiamo con

```
x { s p o }
```

una tripla (s, p, o) che, nell'esempio, ha nome x.

Conjecturing: Conjecturing è la funzione $conj : \mathcal{T} \to \mathcal{T}$ tale che, per ogni tripla RDF $t_1 = (s_1, p, o_1), conj(t_1) = (s, p_{s,o}, o)$ dove:

1. Identità del soggetto: $s_1 = s$.
2. Identità dell'oggetto: $o_1 = o$.
3. Disgiunzione: $\forall s_j, o_k$ tali che $(s_j, p_{s,o}, o_k) \in \mathcal{T}$, abbiamo che $s = s_j$ e $o = o_k$.

Congetture, predicati congetturali, triple congetturali: data la tripla $(s, p, o) \in \mathcal{T}$, la sua congettura è la tripla:

$$conj((s, p, o)) = (s, p_{s,o}, o).$$

I *predicati congetturali* (o *weak predicate*) di un predicato p sono tutti i predicati che appartengono all'insieme $\mathcal{C}onj_p$, così definito:

$$\mathcal{C}onj_p = \{cp \in \mathcal{I} | \exists s \in S_p, o \in O_p, conj((s, p, o)) = (s, cp, o)\}$$

.

Teorema 1: ogni predicato congetturale viene usato in solo una tripla.

Dimostrazione: deriva dal punto 3 della definizione 1 (Disgiunzione): Date due triple $(s_1, p_{s,o}, o_1), (s_2, p_{s,o}, o_2) \in \mathcal{T}$.

Dal punto 3 della definizione 1 (Disgiunzione), abbiamo che $s_1 = s_2$ e $o_1 = o_2 . \forall (s, p, o) \in \mathcal{T}, \exists! p_{s,o}$ tale che $conj((s, p, o)) = (s, p_{s,o}, o)$.

Corollario 1: la funzione $conj$ è unica (salvo modifiche al nome del predicato).

Dimostrazione: deriva immediatamente dal Teorema 1.

Forma Congetturale: un predicato q è definito *forma congetturale* di p se esiste una coppia di soggetti e oggetti $< s, o >$ tale che $conj((s, p, o)) = (s, q, o)$.

Collasso (funzione): il Collasso è una funzione $collapses : \mathcal{T} \to \mathcal{T}$ tale che, per ogni tripla RDF $t = (s, p, o)$, $collapses(s, p, o) = (s, p_{s,o}, o)$ sse $conj((s, p, o)) = (s, p_{s,o}, o)$, altrimenti non definito.

Predicato collassato; collasso (tripla): sia $(s, p_{s,o}, o)$ una Congettura. Il predicato collassato di $p_{s,o}$ è il predicato p tale per cui

$$collapses(s, p, o) = (s, p_{s,o}, o)$$

. Si definisce *Collasso* la tripla

$$((s, p, o), collapse, (s, p_{s,o}, o))$$

.

4.2 Simple Interpretation di RDF e Congetture

In RDF, una "Simple Interpretation" I è una struttura composta da:

1. un insieme non vuoto IR di risorse, chiamato dominio o universo di I;
2. un insieme IP, ovvero l'insieme delle proprietà di I;
3. una mappatura $IEXT$ da IP al prodotto cartesiano $IR \times IR$, ovvero all'insieme delle coppie $< x, y >$ con x e y in IR;
4. una mappatura IS dagli IRI a IR - al fine di mappare risorse e proprietà;
5. una mappatura parziale IL dai letterali a IR - al fine di mappare i letterali.

$IEXT(p)$ è l'estensione di p, cioè l'insieme delle coppie che rappresentano gli argomenti per i quali la proprietà p è verificata.

Secondo [Pat14], un'estensione semantica di RDF corretta consiste in "un insieme di assunti semantici che diano agli IRI significati aggiuntivi sulla

base di altre specifiche o convenzioni. Quando questo succede, le estensioni semantiche devono conformarsi alle condizioni minime di verità già enunciate, estendendole per accogliere i significati aggiuntivi".

Sulla scorta di questo, estendiamo la Simple Interpretation di RDF aggiungendo un nuovo insieme di proprietà congetturali, IPC, disgiunto dall'insieme delle proprietà IP, dove i predicati congetturali sono creati sul momento.

Aggiungiamo una nuova mappatura $IEXTC$ da IPC al prodotto cartesiano $IR \times IR$. $IEXTC(cp)$ identifica le coppie per le quali la proprietà cp è vera.

A causa della proprietà di Disgiunzione della funzione congetturale $conj$ vista nella sezione 4.1, la coppia che soddisfa la proprietà cp sarà sempre unica.

Abbiamo bisogno di specificare una mappatura aggiuntiva $CONJFORM$ da IP a IPC per mappare le forme congetturali delle proprietà.

Riportiamo in blu gli elementi propri delle Congetture in aggiunta a quelli della specifica RDF.

La nostra *Simple Interpretation* completa di I in RDF con le Congetture è quindi:

1. un insieme non vuoto IR di risorse, chiamato dominio o universo di I;
2. un insieme IP, ovvero l'insieme delle proprietà di I;
3. un insieme IPC, ovvero l'insieme delle proprietà congetturali di I. $IPC \cap IP = \emptyset$;
4. una mappatura $IEXT$ da IP al prodotto cartesiano $IR \times IR$, ovvero all'insieme delle coppie $< x, y >$ con x e y in IR;
5. una mappatura iniettivag $IEXTC$ da IPC a $IR \times IR$, in altre parole all'insieme di coppie $< x, y >$ con $x, y \in IR$. Per definizione di mappatura iniettiva, se $IEXTC(a) = IEXTC(b)$, allora $a = b$, ovvero, ogni $cp \in IPC$ si applica unicamente a una specifica coppia $< x, y >$;

6. una mappatura $CONJFORM$ da IP a IPC per mappare le forme congetturali delle proprietà;

7. una mappatura IS da IRI a IR - per mappare risorse, proprietà e proprietà congetturali;

8. una mappatura parziale IL da letterali a IR - per mappare i letterali;

9. una congettura potrebbe essere rappresentata come un insieme di statement individuali che compongono una congettura. Questo insieme è un *grafo congetturale*

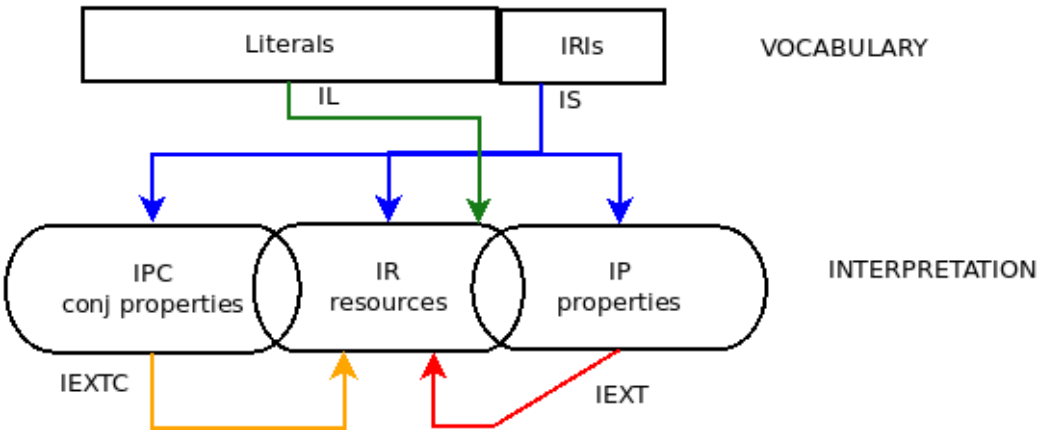

Figura 4.1: Schema del'Interpretazione

4.3 Congetture

4.3.1 Condizioni semantiche

Delineiamo le condizioni semantiche. Come fatto poc'anzi, riporteremo in blu le regole per le Congetture:

- se E è un letterale allora $I(E) = IL(E)$

- se E è un IRI allora $I(E) = IS(E)$
- se E è una tripla senza blank node $s\ p\ o\ .$ allora
  - $I(E) = true$ se $I(p) \in IP$ e
  - la coppia $< I(s), I(o) > \in IEXT(I(p))$
  - altrimenti $I(E) = false$.
- se E è una tripla congetturale $s\ cp\ o\ .$ allora
  - $I(E) = true$ se $I(cp) \in IPC$ e
  - la coppia $< I(s), I(o) > \in IEXTC(I(cp))$
  - $I(cp) \in CONJFORM(I(p))$ con $I(p) \in IP$
  - altrimenti $I(E) = false$.
- se E è un grafo RDF senza blank node allora $I(E) = false$ se $I(E') = false$ per qualsiasi tripla $E' \in E$, altrimenti $I(E) = true$.
- se E è un grafo congetturale senza blank node allora $I(E) = false$ se $I(E') = false$ per qualsiasi tripla $E' \in E$, altrimenti $I(E) = true$.

L'ultima clausola cattura la definizione di grafo congetturale vista precedentemente.

4.3.2 Modello

Dato un grafo RDF G, diciamo che l'Interpretazione I è un modello del grafo G se tutte le triple del grafo G sono soddisfatte, ovvero, se sono vere secondo le regole di I. In questo caso, possiamo dire che l'Interpretazione I *soddisfa* G.

La nozione di modello è alla base del *simple entailment*: seguendo la terminologia standard, possiamo dire che I soddisfa (semplicemente) G quando $I(G) = true$; che G è (semplicemente) soddisfacibile quando esiste una

semplice interpretazione che lo soddisfa, altrimenti è semplicemente insoddisfacibile, e che un grafo E implica semplicemente un grafo G quando ogni interpretazione che soddisfa E soddisfa anche G. Se due grafi G e F si implicano a vicenda, allora sono logicamente equivalenti. [Pat14]

La nostra Interpretazione I è un modello del seguente grafo congetturale?

```
:deVereWroteHamlet {
   :Hamlet conj001:creator :EdwardDeVere .
   conj001:creator conj:isAConjecturalFormOf dc:creator .
}
```

Definiamo gli insiemi:

- $IR = \{dVWH, h, c, cc1, e, iacf\}$
- $IP = \{c, iacf\}$
- $IPC = \{cc1\}$

Le funzioni:

- $IS(: deVereWroteHamlet) \rightarrow dVWH$
- $IS(: Hamlet) \rightarrow h$
- $IS(conj001 : creator) \rightarrow cc1$
- $IS(: EdwardDeVere) \rightarrow e$
- $IS(conj : isAConjecturalFormOf) \rightarrow iacf$
- $IS(dc : creator) \rightarrow c$
- $IEXT(c) = \emptyset$
- $IEXT(iacf) = \{< cc1, c >\}$
- $IEXTC(cc1) = \{< h, e >\}$

- $CONJFORM(c) = cc1$

IEXT(c) è vuoto. Questo è corretto: non c'è ancora nessuna asserzione nel dataset riguardante una proprietà *effettiva* (nel caso sarebbe $dc : creator$), infatti stiamo ancora avendo a che fare con una congettura.

Tutte le clausole sembrano verificate. Possiamo dire che la *Simple Interpretation* (s-interpretation) I è un modello del nostro grafo.

4.4 Collasso alla realtà

Ad un certo punto, potremmo voler dichiarare una congettura vera.

In questo caso dobbiamo specificare un meccanismo per esprimere gli statement della congettura in piena forza, diventando effettivi.

Questo meccanismo è chiamato "collasso alla realtà".

E' composto da esattamente due triple aggiunte al dataset, dove:

- nella prima nuova tripla, viene utilizzata la proprietà "effettiva" invece della corrispondente proprietà congetturale;
- nella seconda tripla si dichiara che la prima nuova tripla "collassa" la congettura. Questa tripla viene chiamata "collasso".

Vengono quindi aggiunte due nuove triple. Il resto del dataset rimane immutato, niente viene rimpiazzato né cancellato. In questo modo possiamo tener traccia di quello che sta succedendo e di tutte le relazioni tra i grafi.

Ragioniamo su un esempio di collasso a realtà:

```
:attr1 {
    :Hamlet conj003:creator :Shakespeare .
    conj003:creator conj:isAConjecturalFormOf dc:creator.
}

:attr1Cot {
```

```
    :Hamlet dc:creator :Shakespeare .
}

:attr1Cot  conj:collapses :attr1 .
```

`:attr1` viene collassato aggiungendo una tripla `:attr1Cot` nella quale il predicato congetturale viene rimpiazzato dal corrispondente predicato effettivo.

La tripla aggiuntiva finale funziona come spiegazione del collasso: `:attr1Cot` collassa la congettura `:attr1` per mezzo della proprietà `conj:collapses`.

Per maggior chiarezza, si tengano a mente le seguenti definizioni:

- la tripla congetturale che viene collassata sarà la "tripla congetturale";
- La nuova tripla che collassa la congettura sarà la "tripla collassante";
- l'ultima tripla sarà la "tripla di collasso".

4.4.1 Interpretazione $[I + COLLAPSE]$

Abbiamo bisogno di estendere ancora una volta la nostra Interpretazione con le Congetture I aggiungendo le regole del Collasso alla Realtà.

Definiamo un nuovo mapping *collapses* da triple a triple che mappa la relazione tra le *triple congetturali* e le rispettive *triple collassanti*:

$collapses(s, p, o) = (s, cp, o)$ iff $conj(s, p, o) = (s, cp, o)$.

Il collasso collassa la tripla congetturale se, e solo se, essa è l'unica congettura della *tripla collassante*.

Possiamo chiaramente vedere che la *tripla di collasso* è la traduzione in RDF della definizione di collasso che abbiamo introdotto nella sezione 4.1, cioè la tripla:

$$((s, p, o), \quad collapses, \quad (s, cp, o))$$

Da un punto di vista più formale, data una tripla congetturale E, aggiungiamo la *tripla collassante* E_{cot} $\{s \quad p \quad o.\}$.

Le condizioni semantiche del collasso alla realtà dovrebbero quindi essere le seguenti:

- sia E una tripla congetturale
  $\{s \quad cp \quad o \quad .\}$

  sia E_{cot} una tripla collassante
  $\{s \quad p \quad o \quad .\}$

  sia $E_{collapse}$ la tripla di collasso
  $\{E_{cot} \quad collapses \quad E\}$

  quindi:

  - $I(E) = true$ se $I(cp) \in IPC$ e
  - $I(E_{cot}) = true$ se $I(p) \in IP$ e
  - $CONJFORM(I(p)) = I(cp)$ e
  - la coppia $< I(s), I(o) > \in IEXTC(I(cp))$ e
  - la coppia $< I(s), I(o) > \in IEXT(I(p))$ e
  - $I(E_{collapse}) = true$ se $collapses(I(E_{cot})) = I(E)$, cioè:
    $collapses(I(s), I(p), I(o)) = (I(s), I(cp), I(o))$
  - altrimenti $I(E) = false$ e $I(E_{cot}) = false$ e $I(E_{collapse}) = false$.

4.4.2 Modello

Ora verifichiamo se la nostra Interpretazione $[I{+}COLLAPSE]$ può essere un modello del nostro grafo di esempio.

Definiamo gli insiemi:

- $IR = \{a1, h, cc3, s, iacf, c, a1cot, cl\}$
- $IP = \{iacf, c, cl\}$

- $IPC = \{cc3\}$

Le funzioni:

- $IS(: attr1) \rightarrow a1$
- $IS(: Hamlet) \rightarrow h$
- $IS(conj003 : creator) \rightarrow cc3$
- $IS(: Shakespeare) \rightarrow s$
- $IS(conj : isAConjecturalFormOf) \rightarrow iacf$
- $IS(dc : creator) \rightarrow c$
- $IS(: attr1Cot) \rightarrow a1cot$
- $IS(conj : collapses) \rightarrow cl$
- $IEXT(iacf) = \{< cc3, c >\}$
- $IEXT(c) = \{< h, s >\}$
- $IEXT(cl) = \{< a1, a1cot >\}$
- $IEXTC(cc3) = \{< h, s >\}$
- $CONJFORM(c) = cc3$

Verifichiamo le condizioni semantiche per il collasso alla realtà:

- $cc3 \in IPC$? Sì;
- $c \in IP$? Sì;
- $CONJFORM(c) = cc3$? Sì;
- la coppia $< h, s > \in IEXTC(cc3)$? Sì;
- la coppia $< h, s > \in IEXTC(c)$? Sì;

- $collapses(h, c, s) = (h, cc3, s)$? Sì, perché $CONJFORM(c) = cc3$, quindi $conj(h, c, s) = (h, cc3, s)$

Sembrano tutte soddisfatte.

Quindi la nostra Interpretazione $[I + COLLAPSE]$ è un modello del nostro grafo.

Come vedremo nella sezione 4.5.2, la *tripla collassante* e la *tripla di collasso* possono essere raggruppate senza problemi all'interno di un *grafo di collasso*:

```
:attr1 {
    :Hamlet conj003:creator :Shakespeare .
    conj003:creator conj:isAConjecturalFormOf dc:creator.
}

:collapseOfattr1 {
    :Hamlet dc:creator :Shakespeare .
    :collapseOfattr1 conj:collapses :attr1 .
}
```

L'ultimo grafo `:collapseOfattr1` contiene infatti entrambe le triple.

4.5 Grafi Congetturali e Grafi di Collasso

4.5.1 Grafi Congetturali

Un *Grafo Congetturale* è una rappresentazione di una congettura per mezzo di un insieme di statement singoli che, insieme, compongono la congettura nella sua interezza.

Tutte le triple all'interno di un *Grafo Congetturale* devono avere sia un predicato congetturale utilizzato esattamente una volta, sia una tripla `conj:isAconjecturalFormOf` che colleghi il predicato congetturale al predicato effettivo originale.

Le condizioni semantiche per i *Grafi Congetturali* sono:

- se E è un letterale allora $I(E) = IL(E)$
- se E è un IRI allora $I(E) = IS(E)$
- se E è una tripla congetturale $s\ cp\ o$. allora
  - $I(E) = true$ se $I(cp) \in IPC$ e
  - la coppia $< I(s), I(o) > \in IEXTC(I(cp))$
  - $I(cp) \in CONJFORM(I(p))$ con $I(p) \in IP$ e
  - E deve essere unico nel grafo
  - altrimenti $I(E) = false$.
- se E è una tripla $cpconj : isAConjecturalFormOfp$ allora
  - $I(E) = true$ se $I(cp) \in IPC$ e
  - $I(p) \in IP$ e
  - $I(cp) \in CONJFORM(I(p))$ e
  - $I(conj : isAConjecturalFormOf) \in IP$
  - la coppia $< I(cp), I(p) > \in IEXT(I(conj : isAConjecturalFormOf))$
  - altrimenti $I(E) = false$
- se E è un grafo congetturale senza blank node allora $I(E) = false$ se $I(E') = false$ per qualche tripla $E' \in E$, altrimenti $I(E) = true$.

4.5.2 Grafi di Collasso

I *Grafi di Collasso* sono dei grafi che si collegano esplicitamente con una congettura e rendono possibile tracciare quale congettura sia stata collassata.

Devono essere composti come minimo da:

- la "forma effettiva" della congettura da collassare;
- una tripla `conj:collapse` (ovvero la *tripla di collasso* vista precedentemente) che collega la congettura al suo collasso.

Le condizioni semantiche sono le seguenti:

- sia E_{conj} una tripla congetturale $s\ cp\ o\ .$
- sia $E_{collapse}$ un grafo di collasso
- se E è un letterale allora $I(E) = IL(E)$
- se E è un IRI allora $I(E) = IS(E)$
- se E è una tripla $s\ p\ o\ .$ allora
  - $I(E) = true$ se $I(p) \in IP$ e
  - la coppia $< I(s), I(o) > \in IEXT(I(p))$
  - $CONJFORM(I(p)) = I(cp)$ dove cp è il predicato congetturale della congettura da collassare
  - altrimenti $I(E) = false$.
- se E è una tripla $E_{collapse} \quad conj : collapses \quad E_{conj}$ allora
  - $I(E) = true$ se $I(conj : collapses) \in IP$ e
  - la coppia $< I(E_{collapse}), I(E_{conj}) > \in IEXT(I(conj : collapses))$
  - altrimenti $I(E) = false$
- se E è un grafo congetturale senza blank node allora $I(E) = false$ se $I(E') = false$ per qualunque tripla $E' \in E$, altrimenti $I(E) = true$.

4.6 Blank Node

Ad un certo punto potremmo voler esprimere, con le Congetture, concetti ancora più "incerti", magari aventi a che fare con entità senza nome o con valori non specificati.

In RDF questo è possibile tramite i *blank nodes*, che indicano l'esistenza di un'entità senza utilizzare un IRI per identificarne una specifica.

In questa sezione, come esempio pratico, utilizzeremo una semplice frase che però comporta l'utilizzo di un blank node, ovvero: "Mu'ammar Gheddafi ha dichiarato che l'autore di Otello era qualcuno di nazionalità Araba". [1]

In questo caso specifico non sappiamo chi sia questo "qualcuno", e non possiamo identificarlo con un IRI.

In ogni caso, le informazioni che stiamo comunicando con la nostra congettura mantengono comunque un certo significato, e rappresentano sicuramente una base sulla quale effettuare ragionamenti.

Per riportare la frase di Gheddafi nello stile dei nostri esempi, potremmo riformularla così:

"E' stato ipotizzato che Otello sia stato scritto da qualcuno, e che questo qualcuno fosse di nazionalità Araba. Questa affermazione è stata attribuita a Mu'ammar Gheddafi".

In RDF sarebbe così:

```
:ArabWroteOthello {
  :Othello conj002:creator _:z .
  conj002:creator conj:isAConjecturalFormOf dc:creator .
  _:z dbpedia-owl:nationality :Arab .}

:ArabWroteOthello prov:wasAttributedTo :MalQaddafi .
```

Viene piuttosto naturale considerare il termine "qualcuno" come un blank node.

4.6.1 Interpretazione $[I + A]$

Per gestire i blank node, dobbiamo aggiungere un nuovo mapping A dai blank node a IR.

Pertanto, estendiamo la nostra Interpretazione I:

- $[I + A](x) = I(x)$ per i nomi

[1] Gheddafi l'ha veramente dichiarato nel dicembre 1988, vedi [Bha06]

- $[I + A](x) = A(x)$ se x è un blank node.

Aggiungiamo alla nostra Interpretazione un paio di condizioni semantiche per i blank node. Una è la condizione semantica "standard" dei blank node nei grafi RDF, l'altra è specifica per le Congetture:

- se E è un grafo RDF allora $I(E) = true$ se $[I + A](E) = true$ per una mappatura A dall'insieme dei blank node di E a IR, altrimenti $I(E) = false$;
- se E è un grafo congetturale allora $I(E) = true$ se $[I+A](E) = true$ per una mappatura A dall'insieme dei blank nodes di E a IR, altrimenti $I(E) = false$.

4.6.2 Modello

La nostra Interpretation $[I + A]$ con i blank node è un modello del nostro grafo di esempio?

Ragioniamo solo sulla parte `:ArabWroteOthello`.

Sia A la nostra mappatura dei blank node su IR: $\_ : z \rightarrow zz$.

La nostra Interpretazione $[I + A]$ sarà:

- $IR = \{awo, o, c, cc2, iacf, n, a, zz\}$
- $IP = \{c, iacf, n\}$
- $IPC = \{cc2\}$

Le funzioni:

- $IS(: ArabWroteOthello) \rightarrow awo$
- $IS(: Othello) \rightarrow o$
- $IS(conj002 : creator) \rightarrow cc2$
- $IS(dc : creator) \rightarrow c$

- $IS(dbpedia - owl : nationality) \rightarrow n$
- $IS(: Arab) \rightarrow a$
- $IS(conj : isAConjecturalFormOf) \rightarrow iacf$
- $IEXT(c) = \emptyset$
- $IEXT(iacf) = \{< cc2, c >\}$
- $CONJFORM(c) = cc2$
- $IEXT(n) = \{< zz, a >\}$
- $IEXTC(cc2) = \{< o, zz >\}$

Anche in questo caso `IEXT(c)` è vuota perché non ci sono ancora asserzioni riguardanti la proprietà `dc:creator`, infatti la congettura rimane ancora una congettura.

Le ultime due funzioni sono verificate grazie alla mappatura A dei nostri blank node.

Tutte le clausole sono verificate. La nostra Interpretazione $I + A$ è un modello del nostro grafo.

Questo approccio ci permette di ragionare con i blank node potendo generare scenari nei quali potremmo definire la mappatura A da blank node a specifiche risorse a nostra scelta, esplorando quindi le nuove relazioni che potrebbero sorgere tra le triple.

4.7 Applicazioni ulteriori delle Congetture

4.7.1 Congettura di una Congettura - Annidamento di Congetture

Immaginiamo adesso di voler esprimere una congettura di un'altra congettura. Naturalmente, il processo può coinvolgere tutte le congetture di congetture che vogliamo.

Ad esempio:

```
:conjecture01 {
    :Hamlet conj004:creator :EdwardDeVere .
    conj004:creator conj:isAConjecturalFormOf dc:creator .
}

:conjecture02 {
 :conjecture01 conj004:wasAttributedTo :JThomasLooney .
 conj004:wasAttributedTo conj:isAConjecturalFormOf prov:wasAttributedTo .
}

:conjecture03 {
:conjecture02 conj004:wasInformedBy <https://bit.ly/3wSH76A> .
conj004:wasInformedBy conj:isAConjecturalFormOf prov:wasInformedBy .
}

:conjecture03 prov:wasAttributedTo :FabioVitali .
```

Qui abbiamo tre Congetture differenti, una che diventa l'oggetto dell'altra:

- la prima dice che si è ipotizzato che Amleto sia stato scritto da Edward De Vere;
- la seconda dice che si pensa che la prima congettura possa essere stata fatta da J. Thomas Looney;
- la terza dice che si è ipotizzato che la seconda congettura possa essere stata portata all'attenzione del grande pubblico da una risorsa con URL `https://bit.ly/3wSH76A`.

La quarta tripla, che non è una congettura, dice che l'ultima congettura è stata attribuita a Fabio Vitali.

Rileggendo il tutto (più o meno) all'indietro: Fabio Vitali ha affermato che la risorsa all'url `https://bit.ly/3wSH76A` potrebbe aver portato alla luce la possibilità che J. Thomas Looney possa aver detto che si ipotizza che Amleto possa essere stato scritto da Edward De Vere.

Potremmo immaginarlo come una pila, o una scala, dove, a livello 0, abbiamo la prima Congettura, poi, mentre saliamo di livello nella scala, le congetture che troviamo sono costruite con le congetture del livello immediatamente inferiore come soggetto (o oggetto, o entrambi):

```
Livello 2:            E{3}={E{2}, cp3, o3}
Livello 1:                  E{2}=(E{1}, cp2, o2}
Livello 0:                        E{1}=(s1, cp1, o1)
```

Sviluppando il Livello 2, si può vedere che sono annidate le une nelle altre: $E3 = (E1, cp2, o2), cp3, o3) = (((s1, cp1, o1), cp2, o2), cp3, o3)$

Interpretatione $[I + NESTEDCONJ]$

Per delineare le regole per le Congetture di Congetture (o Congetture annidate), dobbiamo ragionare con coppie di Congetture.

Come affermato precedentemente, le Congetture a livelli > 0 potrebbero avere le congetture di livello inferiore come loro soggetto, oggetto o entrambi.

Per brevità, per il momento ci limitiamo a ragionare con il caso delle congetture di livello inferiore come soggetto.

1. se E_0 è una Congettura senza blank node (s_0, cp_0, o_0), $E_1 = (E_0, cp_1, o_1)$
2. se E_1 è una Congettura senza blank node (E_0, cp_1, o_1), $E_2 = (E1, cp_2, o_2)$
   [...]
3. se E_{k-1} è una Congettura senza blank node di "livello più alto" $(E_{k-2}, cp_{k-1}, o_{k-1})$, $E_k = (E_{k-1}, cp_k, o_k)$
4. se E_k è una Congettura senza blank node di "livello più alto" (E_{k-1}, cp_k, o_k), $E_{k+1} = (E_k, cp_{k+1}, o_{k+1})$

Dovremmo trovarci con i casi seguenti:

- caso base:
  - $E_0 = (s_0, p_0, o_0)$
- Primo caso:
  - $E_1 = (E_0, cp_1, o_1)$
  - $E_1 = (s_1, cp_1, E_0)$
- k-esimo caso:
  - $E_k = (E_{k-1)}, cp_k, o_k)$
  - $E_k = (s_k, cp_k, E_{k-1})$

Estendiamo la nostra Interpretazione I aggiungendo nuove regole.

L'estensione sarà suddivisa in casi, a seconda del tipo di Congettura da valutare.

Dobbiamo anche imporre un ordine alle operazioni: cominciamo dalla (dalle) Congettura (Congetture) a livello "più basso", cioè quella (quelle) che non coinvolgono altre congetture, la (le) valutiamo e "saliamo" al gradino successivo della "scala".

Quindi, l'estensione all'Interpretazione I sarà:
Caso base - per le Congetture a livello 0:

- Sia E_0 una tripla congetturale
  $\{s_0 \quad cp_0 \quad o_0 \quad .\}$ allora
  - $I(E_0) = true$ se $I(cp_0) \in IPC$ e
  - la coppia $< I(s_0), I(o_0) > \in IEXTC(I(cp_0))$
  - altrimenti $I(E_0) = false$

"Salendo" la "scala" e giungendo alla Congettura E_k, possiamo fare affidamento sul fatto che la Congettura E_{k-1} è già stata verificata dai passi

precedenti.

k-esimo Caso S - per Congetture di livello maggiore di 0 (da 1 in su) aventi un'altra Congettura come soggetto:

- sia E_{k-1} una tripla congetturale

  sia E_k una tripla congetturale
  $\{E_{k-1} \quad cp_k \quad o_k \quad .\}$

  allora

  - $I(E_k) = true$ se $I(E_{k-1}) = true$ e
  - $I(cp_k) \in IPC$ e
  - la coppia $< I(E_{k-1}), I(o_k) > \in IEXTC(I(cp_k))$
  - altrimenti $I(E_k) = false$

k-esimo Caso O - - per Congetture di livello maggiore di 0 (da 1 in su) aventi un'altra Congettura come oggetto:

- sia E_{k-1} una tripla congetturale

  sia E_k una tripla congetturale
  $\{s_k \quad cp_k \quad E_{k-1} \quad .\}$

  allora

  - $I(E_k) = true$ se $I(E_{k-1}) = true$ e
  - $I(cp_k) \in IPC$ e
  - la coppia $< I(s_k), I(E_{k-1}) > \in IEXTC(I(cp_k))$
  - altrimenti $I(E_k) = false$

k-esimo Case SO - per Congetture di livello maggiore di 0 (da 1 in su) aventi un'altra Congettura come soggetto e un'altra ancora come oggetto:

- sia $E_{(k-1)a}$ una tripla congetturale

  sia $E_{(k-1)b}$ una tripla congetturale

  sia E_k una tripla congetturale
  $\{E_{(k-1)a} \quad cp_k \quad E_{(k-1)b} \quad .\}$

  allora

  - $I(E_k) = true$ se $I(E_{(k-1)a}) = true$ e
  - $I(E_{(k-1)b}) = true$ e
  - $I(cp_k) \in IPC$ e
  - la coppia $< I(E_{(k-1)a}), I(E_{(k-1)b}) > \in IEXTC(I(cp_k))$
  - altrimenti $I(E_k) = false$

Modello

La nostra Interpretazione $[I + NESTEDCONJ]$ è un modello dell'esempio di congetture annidate visto poc'anzi?

Definiamo gli insiemi:

- $IR = \{c1, h, cc4, edv, iacf, c, c2, cwa4, jtl, pwa, c3, cwib4, http, pwib, fv\}$
- $IP = \{c, iacf, pwa, pwib\}$
- $IPC = \{cc4, cwa4, cwib4\}$

Le funzioni:

- $IS(: conjecture01) \rightarrow c1$
- $IS(: Hamlet) \rightarrow h$
- $IS(conj004 : creator) \rightarrow cc4$
- $IS(: EdwardDeVere) \rightarrow edv$
- $IS(conj : isAConjecturalFormOf) \rightarrow iacf$

- $IS(dc : creator) \rightarrow c$
- $IS(: conjecture02) \rightarrow c2$
- $IS(conj004 : wasAttributedTo) \rightarrow cwa4$
- $IS(: JThomasLooney) \rightarrow jtl$
- $IS(prov : wasAttributedTo) \rightarrow pwa$
- $IS(: conjecture03) \rightarrow c3$
- $IS(conj004 : wasInformedBy) \rightarrow cwib4$
- $IL(< https://bit.ly/3wSH76A >) \rightarrow http$
- $IS(prov : wasInformedBy) \rightarrow pwib$
- $IS(: FabioVitali) \rightarrow fv$
- $IEXT(iacf) = \{< cc4, c >, < cwa4, pwa >, < cwib4, pwib >\}$
- $IEXT(c) = \emptyset$
- $IEXT(pwa) = \{< c3, fv >\}$
- $IEXT(pwib) = \emptyset$
- $IEXTC(cc4) = \{< h, ev >\}$
- $IEXTC(cwa4) = \{< c1, jtl >\}$
- $IEXTC(cwib4) = \{< c2, http >\}$
- $CONJFORM(c) = cc4$
- $CONJFORM(cwa4) = pwa$
- $CONJFORM(cwib4) = pwib$

Controlliamo ora la validità delle regole della nuova estensione semantica.

Dobbiamo partire dalla Congettura che non dipende da altre Congetture, ovvero da `:conjecture01`, e usiamo il "Caso Base":

- $cc4 \in IPC$? Sì
- la coppia $< h, ev > \in IEXTC(cc4)$ S'i

La congettura "di base" sembra verificata. Possiamo dire che $c1 = true$.

Ora "saliamo la scala" fino a `:conjecture02`. Visto che si basa su un'altra Congettura (`:conjecture01`, già verificata) stante come suo soggetto, useremo il "k-esimo Caso S"

- $c1 = true$? Sì
- $cwa4 \in IPC$? Sì
- la coppia $< c1, jtl > \in IEXTC(cwa4)$? Sì

Possiamo quindi dire che anche $c2 = true$.

Saliamo di un altro "scalino". `:conjecture03` ha `:conjecture02` come suo soggetto. Usiamo ancora una volta il "k-esimo Caso S"

- $c2 = true$? Sì
- $cwib4 \in IPC$? Sì
- la coppia $< c2, http > \in IEXTC(cwib4)$? Sì

$\rightarrow c3 = true$.

L'ultima tripla:

```
:conjecture03 prov:wasAttributedTo :FabioVitali
```

è soddisfatta dalle regole dell'Interpretazione semplice I.
Sembra che tutto sia verificato.

Possiamo quindi dire che la nostra Interpretatione $[I+NESTEDCONJ]$ soddisfa tutte le triple del grafo.

4.7.2 Congettura di un collasso

Consideriamo la seguente frase:

> Secondo l'Enciclopedia Britannica (`https://bit.ly/3qgakFT`), Samuel Johnson ha attribuito la paternità di Amleto a William Shakespeare, e questa attribuzione è corretta.

Questa frase è più di un semplice collasso: è una congettura di un articolo dell'Enciclopedia Britannica che a) attribuisce a Samuel Johnson una congettura (riguardo alla paternità di Amleto) e b) esprime totale fiducia in tale congettura (ovvero la collassa).

La sua rappresentazione è:

```
:attribution01 {
  :Hamlet conj005:creator :WilliamShakespeare .
  conj005:creator conj:isAConjecturalFormOf dc:creator .
}

:collapseOfAttribution01 {
  :attribution01 conj005:wasAttributedTo :SamuelJohnson .
  conj005:wasAttributedTo conj:isAConjecturalFormOf prov:wasAttributedTo .

  :collapseOfAttribution01 conj005:collapses :attribution01 .
  conj005:collapses conj:isAConjecturalFormOf conj:collapses .
}

:collapseOfattribution01 prov:wasInformedBy <https://bit.ly/3qgakFT> .
```

In questo caso abbiamo una Congettura di un collasso che, qualora collassasse, comporterebbe, come effetto collaterale, l'attivazione del collasso di un'altra Congettura, asserendola pienamente.

Si noti che, nel suo stato corrente e non collassato, il tutto è ancora congetturato.

Possiamo vedere il predicato (ormai familiare) *collapses* nella sua forma congetturale, non ancora quindi nella sua forma effettiva. Questo predicato, una volta che lo si incontrerà nella sua forma effettiva, scatenerà il collasso del suo oggetto.

Lo vedremo nella prossima sezione.

4.7.3 Collassi a cascata

I collassi a cascata sono tutti i collassi che occorrono ricorsivamente quando Congetture di collassi multiple e annidate collassano.

Per esempio, se volessimo dichiarare che l'elemento `:collapseOfAttribution01` è vero, dovremmo collassarlo aggiungendo la *tripla collassante* e la *tripla di collasso* seguendo le regole dell'Interpretazione `[I` COLLAPSE]+ viste precedentemente in 4.4.1.

Per semplicità, facciamo ricorso alla nozione di grafo di collasso (vedi 4.5.2), secondo la quale possiamo raggruppare le triple aggiuntive all'interno del grafo di collasso `:collapseOfcollapseOfAttribution01`.

Ora che la nostra congettura di un collasso è dichiarata vera, quindi il collasso è vero ed effettivo, dobbiamo attivarlo, collassando `:attribution01`.

Generalizzando, una volta che una Congettura di un collasso è dichiarata vera, dobbiamo attivare una nuova regola speciale per il suo collasso alla realtà:

Ogni volta che il predicato `collapses` viene incontrato nella sua forma effettiva all'interno di un grafo di collasso, il suo oggetto, qualora sia una congettura, deve essere collassato.

Se l'oggetto è una tripla congetturale $scpo$, dobbiamo aggiungere la *tripla collassante* e la *tripla di collasso*.

Tornando al nostro esempio, possiamo vedere che la *tripla di collasso* è già definita nella sua forma effettiva. Infatti abbiamo:

```
:collapseOfAttribution01 conj:collapses :attribution01 .
```

Quindi abbiamo solo bisogno di aggiungere la *tripla collassante*, asserendo il contenuto della Congettura in piena forza, all'interno del grafo dove la proprietà `conj:collapses` è incontrata nella sua forma effettiva.

Il risultato sarà il seguente:

```
:attribution01 {
  :Hamlet conj005:creator :WilliamShakespeare .
  conj005:creator conj:isAConjecturalFormOf dc:creator .
}

:collapseOfAttribution01 {
  :attribution01 conj005:wasAttributedTo :SamuelJohnson .
  conj005:wasAttributedTo conj:isAConjecturalFormOf prov:wasAttributedTo .

  :collapseOfAttribution01 conj005:collapses :attribution01  .
  conj005:collapses conj:isAConjecturalFormOf conj:collapses .
}
:collapseOfattribution01 prov:wasInformedBy <https://bit.ly/3qgakFT> .

:collapseOfcollapseOfAttribution01  {
  :attribution01 prov:wasAttributedTo :SamuelJohnson .

  :collapseOfAttribution01 conj:collapses :attribution01 .

  :collapseOfcollapseOfAttribution01 conj:collapses
  :collapseOfAttribution01 .

  :Hamlet dc:creator :WilliamShakespeare.
```

```
}
```

Viene costruito un grafo di collasso per il primo collasso della congettura del collasso, poi, alla fine, viene aggiunta la tripla "effettiva" finale.

A questo punto abbiamo bisogno di estendere l'Interpretazione $[I{+}COLLAPSE]$ vista precedentemente in 4.4.1.

Dopo il collasso della congettura di collasso seguendo le regole di $[I + COLLAPSE]$, troveremo la proprietà `collapses` nella sua forma effettiva all'interno del grafo di collasso.

Dobbiamo estendere la nozione di `collapses` definendola da grafi di collasso a grafi congetturali: $collapses(G) = (E)$ sse $\forall(s, cp, o) \in E, conj(s, p, o) = (s, cp, o)$ con $(s, p, o) \in G$.

Se l'oggetto della proprietà `collapses` è una tripla, che può essere considerata un caso speciale di un grafo, aggiungeremo la sua forma effettiva al grafo congetturale che ospita la proprietà `collapses`.

Se l'oggetto della proprietà `collapses` è un grafo congetturale, dobbiamo aggiungere la forma effettiva di tutte le sue Congetture.

Una cosa importante da tenere in considerazione è che il soggetto della proprietà `collapses` non viene interessato da alcuna operazione: la forma effettiva della congettura (o delle congetture) dell'oggetto di `collapses` è aggiunta (o sono aggiunte) all'*attuale* grafo di collasso indipendentemente dal soggetto.

Nelle condizioni semantiche che seguono applicheremo una semplificazione per maggiore chiarezza adottando la nozione di grafo generico come soggetto teorico della proprietà `collapses`. Il grafo potrà essere più di uno nel caso in cui i soggetti delle Congetture che sono oggetti di `collapses` siano distinti (come nell'esempio precedente). Anche le triple o i grafi congetturali potranno essere più di uno.

Le condizioni semantiche sono:

triple congetturali: Se abbiamo una tripla congetturale come oggetto di `collapses`:

- sia E una tripla congetturale
  $\{s \quad cp \quad o \quad .\}$
  sia E_g un grafo generico
  sia E_{cg} un grafo di collasso
  $\{E_g \quad conj : collapses \quad E$
  $s \quad p \quad o.\}$
  allora
  - $I(E) = true$ se $I(cp) \in IPC$ e
  - $I(E_{cg}) = true$ se $I(p) \in IP$ e
  - $CONJFORM(I(p)) = I(cp)$ e
  - la coppia $< I(s), I(o) > \in IEXTC(I(cp))$ e
  - la coppia $< I(s), I(o) > \in IEXT(I(p))$ e
  - $I(conj : collapses) \in IP$ e
  - la coppia $< I(E_g), I(E > \in IEXT(I(conj : collapses))$
  - altrimenti $I(E) = false$ e $I(E_{cg}) = false$.

grafo congetturale: Se abbiamo un grafo congetturale come oggetto di `conj:collapses`:

- sia E un grafo congetturale
  $\{s_0 \quad cp_0 \quad o_0 \quad .$
  [...]
  $s_k \quad cp_k \quad o_k \quad .\}$
  sia E_g un grafo generico
  sia E_{cg} un grafo di collasso
  $\{s_0 \quad p_0 \quad o_0.$
  [...]
  $s_k \quad p_k \quad o_k \quad .$
  $E_g \quad conj : collapses \quad E\}$
  allora

- $I(E) = true$ if $I(cp_k) \in IPC \ \forall \ cp_k \in E$ e
- $I(E_{cg}) = true$ if $I(p_k) \in IP \ \forall \ p_k \in E_{cg}$ e
- $CONJFORM(I(p_k)) = I(cp_k) \ \forall \ p_k \in E$ e
- qualsiasi coppia $< I(s_k), I(o_k) > \in IEXTC(I(cp_k))$ e
- qualsiasi coppia $< I(s_k), I(o_k) > \in IEXT(I(p_k))$ e
- $I(conj : collapses) \in IP$ e
- la coppia $< I(E_g), I(E) > \in IEXT(I(conj : collapses))$
- altrimenti $I(E) = false$ e $I(E_{cg}) = false$.

Modello

Verifichiamo se l'Interpretazione può essere un modello per il nostro ultimo esempio.

Definiamo gli insiemi:

- $IR = \{a1, h, cc5, ws, iacf, c, coa1, c5wat, sj, pwat, c5cl, cl, pwib, url, ccoa1\}$
- $IP = \{c, iacf, pwat, pwib, cl\}$
- $IPC = \{cc5, c5wat, c5cl\}$

Le funzioni:

- $IS(: attribution01) \rightarrow a1$
- $IS(: Hamlet) \rightarrow h$
- $IS(conj005 : creator) \rightarrow cc5$
- $IS(: WilliamShakespeare) \rightarrow ws$
- $IS(conj : isAConjecturalFormOf) \rightarrow iacf$
- $IS(dc : creator) \rightarrow c$
- $IS(: collapseOfAttribution01) \rightarrow coa1$

- $IS(conj005 : wasAttributedTo) \rightarrow c5wat$
- $IS(: SamuelJohnson) \rightarrow sj$
- $IS(prov : wasAttributedTo) \rightarrow pwat$
- $IS(conj005 : collapses) \rightarrow c5cl$
- $IS(conj : collapses) \rightarrow cl$
- $IS(prov : wasInformedBy) \rightarrow pwib$
- $IL(< https://bit.ly/3qgakFT >) \rightarrow url$
- $IS(: collapseOfcollapseOfAttribution01) \rightarrow ccoa1$
- $IEXT(c) = \{< h, ws >\}$
- $IEXT(iacf) = \{< cc5, c >, < c5wat, pwat >, < c5cl, cl >\}$
- $IEXT(pwat) = \{< a1, sj >\}$
- $IEXT(cl) = \{< a1, coa1 >, < coa1, ccoa1 >\}$
- $IEXT(pwib) = \{< coa1, url >\}$
- $IEXTC(cc5) = \{< h, ws >\}$
- $IEXTC(c5wat) = \{< a1, sj >\}$
- $IEXTC(c5cl) = \{< a1, coa1 >\}$
- $CONJFORM(c) = cc5$
- $CONJFORM(pwat) = c5wat$
- $CONJFORM(cl) = c5cl$

Verifichiamo ora le regole. Ci focalizziamo in particolare sui collassi a cascata.

Dobbiamo procedere a passi. Il primo `conj:collapses` nel nostro grafo di collasso finale ha come oggetto `:attribution01`.

Possiamo considerare `:attribution01` come una singola tripla, visto che contiene solo una Congettura:

- $cc5 \in IPC$? Sì;
- $c \in IP$? Sì;
- $CONJFORM(c) = cc5$? Sì;
- la coppia $< h, ws > \in IEXTC(cc5)$? Sì;
- la coppia $< h, ws > \in IEXT(c)$? Sì;
- $cl \in IP$? Sì;
- la coppia $< a1, coa1 > \in IEXT(cl)$? Sì.

Tutte le condizioni sembrano soddisfatte.

Andiamo al secondo `conj:collapses` del nostro grafo di collasso finale. Presenta `:collapseOfAttribution01` come suo oggetto. Si tratta di un grafo. Applichiamo quindi le regole per i grafi:

- $c5wat, c5cl \in IPC$? Sì;
- $pwat, cl \in IP$? Sì;
- $CONJFORM(c5wat) = pwat; CONJFORM(c5cl) = cl$? Sì;
- $< a1, sj > \in IEXTC(c5wat)$? Sì; $< a1, coa1 > \in IEXTC(c5cl)$? Sì;
- $< a1, sj) > \in IEXT(pwat)$? Sì; $< a1, coa1 > \in IEXT(cl)$? Sì;
- $cl \in IP$? Sì;

- la coppia $< a1, coa1) > \in IEXT(cl)$? Sì.

Si può verificare facilmente che tutte le altre triple dell'esempio possono essere soddisfatte dall'Interpretazione $[I + COLLAPSE]$

La nostra Interpretazione è un modello del grafo.

4.8 Valutazioni

In questo capitolo siamo partiti dal modello formale teorico-insiemistico delineando le caratteristiche di base delle Congetture. Abbiamo poi esteso la *Simple Intepretation* di RDF aggiungendo gli insiemi, le funzioni e le regole semantiche di base delle Congetture. Abbiamo poi esteso ulteriormente il modello semantico approfondendo le caratteristiche delle Congetture sia per quanto riguarda la loro struttura che per quanto riguarda le loro funzionalità, estendendo opportunamente l'*Interpretation*. Ad ogni passo abbiamo verificato il modello, dimostrando che l'estensione soddisfa i grafi dei casi esemplificativi, senza eccezioni. Possiamo quindi affermare che le Congetture sono un'**estensione lecita di RDF**.

Conclusioni

In questa dissertazione è stato presentato un approccio nuovo al problema di esprimere in RDF informazioni senza un valore di verità ben definito, ovvero le Congetture.

Dopo l'introduzione al problema, si è presentata una rassegna delle attuali tecnologie che, con diversi gradi di successo, hanno cercato di implementare la possibilità di effettuare affermazioni su altri statement (ad esempio per le informazioni di provenienza) o di poter gestire informazioni ipotetiche o contrastanti tra loro. Questa analisi ha evidenziato come nessuna di queste tecnologie raggiunga lo stesso potere espressivo delle Congetture. Quelle che ci si avvicinano di più (*Relazioni n-arie*, *N3*), comportano la necessità di gestire una grande complessità strutturale, che si riflette anche nella complessità delle query per interrogare i dati.

Le Congetture, invece, si basano su un approccio ben conosciuto, i *Named Graph*. Aggiungendo solo pochi elementi sintattici e semantici, riescono ad essere trasposte in RDF puro senza perdita di potere espressivo, rimanendo pienamente compatibili con SPARQL, e permettendo la costruzione di query efficaci e relativamente semplici.

Per lo sviluppo della semantica formale delle Congetture, siamo partiti dal modello e dalla *Simple Interpretation* di RDF 1.1. Abbiamo esplorato le caratteristiche chiave delle Congetture verificando la loro compatibilità con la semantica formale di RDF 1.1, estendendola quando necessario. Il risultato, pienamente consistente con RDF, è un'ulteriore riprova della bontà della proposta.

Le Congetture permettono di esprimere concetti il cui valore di verità non è conosciuto, senza però asserirli. Questa è una caratteristica che al momento manca in RDF.

Con le Congetture, mediante una estensione minima del modello, è possibile aggiungere a RDF 1.1 questa potente funzionalità.

L'inclusione delle Congetture all'interno del modello di RDF 1.1 apre una serie di scenari potenzialmente interessanti grazie alle caratteristiche proprie di RDF delineate all'inizio della presente dissertazione, tra le quali, la facile interpretazione da parte di sistemi automatici ed il supporto a sistemi di inferenza.

Sviluppi futuri al presente lavoro potrebbero consistere nell'applicazione del paradigma delle Congetture a dataset RDF reali, permettendo di valutarne sperimentalmente l'effettiva usabilità e complessità. Nel lavoro di Hernandez [HHK15], viene fatta un'analisi comparata, in termini di efficienza di storage e di reperimento delle informazioni via query SPARQL, degli approcci al supporto delle informazioni di provenance da parte di *Reification*, *Relazioni n-arie*, *Singleton Properties*, *Named Graphs*. Questi ultimi costituiscono le basi sulle quali le Congetture sono create. Sarebbe senz'altro interessante applicare la medesima metodologia alle Congetture, verificandone la performance in termini di tempo di esecuzione delle query e surplus di spazio occupato rispetto ai *Named Graphs*.

Ulteriori sviluppi potrebbero riguardare l'implementazione delle Congetture all'interno dei framework RDF esistenti. In questo modo, il formalismo delle Congetture, applicato a casi notevoli in varie discipline scientifiche, permetterebbe di descrivere in RDF, ad esempio, ipotesi contrastanti o dispute su attribuzioni, rendendo finalmente possibile l'utilizzo degli strumenti di RDF per esplorare il mondo dell'incertezza, delle ipotesi e dei dubbi.

Bibliografia

[AW19] Dörthe Arndt e William Van Woensel. «Towards Supporting Multiple Semantics of Named Graphs Using N3 Rules». In: *Proceedings of the 13th RuleML+RR 2019 Doctoral Consortium and Rule Challenge, Bolzano, Italy, September 16-24, 2019*. Vol. 2438. CEUR Workshop Proceedings. Aachen: CEUR-WS.org, 2019. URL: http://ceur-ws.org/Vol-2438/paper6.pdf.

[Arn+14] Dörthe Arndt et al. *Notation3*. Lug. 2014. URL: https://w3c.github.io/N3/spec/.

[Arn+22] Dörthe Arndt et al. *RDF-star and SPARQL-star*. Mar. 2022. URL: https://w3c.github.io/rdf-star/cg-spec/editors_draft.html.

[Bek+21] Chryssoula Bekiari et al. *Definition of the CIDOC Conceptual Reference Model, Version 7.2*. Set. 2021. URL: http://www.cidoc-crm.org/sites/default/files/cidoc_crm_version_7.2.pdf.

[Ber+08] Tim Berners-Lee et al. «N3Logic: A logical framework for the World Wide Web». In: *Theory Pract. Log. Program.* 8.3 (2008), pp. 249–269. DOI: 10.1017/S1471068407003213. URL: https://doi.org/10.1017/S1471068407003213.

[Bha06] J. Bhattacharyya. *William Shakespeare's Othello*. Atlantic critical studies. Atlantic, 2006. ISBN: 9788126904747.

[Car+05] Jeremy J. Carroll et al. «Named graphs, provenance and trust». In: *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*. A cura di Allan Ellis e Tatsuya Hagino. ACM, 2005, pp. 613–622. DOI: 10.1145/1060745.1060835. URL: https://doi.org/10.1145/1060745.1060835.

[HT14] Olaf Hartig e Bryan Thompson. «Foundations of an Alternative Approach to Reification in RDF». In: *CoRR* abs/1406.3399 (2014). arXiv: 1406.3399. URL: http://arxiv.org/abs/1406.3399.

[Haw11] Sandro Hawke. *ISSUE-25: Should we deprecate (RDF 2004) reification of statements?* Apr. 2011. URL: https://www.w3.org/2011/rdf-wg/track/issues/25.

[Haw+13] Sandro Hawke et al. *SPARQL 1.1 Entailment Regimes*. W3C Recommendation. W3C, mar. 2013. URL: https://www.w3.org/TR/sparql11-entailment/.

[HHK15] Daniel Hernández, Aidan Hogan e Markus Krötzsch. «Reifying RDF: What Works Well With Wikidata?» In: *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems, Bethlehem, PA, USA, October 11, 2015*. Vol. 1457. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 32–47. URL: http://ceur-ws.org/Vol-1457/SSWS2015%5C_paper3.pdf.

[LWC14] Markus Lanthaler, David Wood e Richard Cyganiak. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation . https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. W3C, 2014.

[NBS14] Vinh Nguyen, Olivier Bodenreider e Amit P. Sheth. «Don't like RDF reification?: making statements about statements using singleton property». In: *23rd International World Wide Web Con-*

ference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014. A cura di Chin-Wan Chung et al. ACM, 2014, pp. 759–770. DOI: 10.1145/2566486.2567973. URL: https://doi.org/10.1145/2566486.2567973.

[NH17] Franco Niccolucci e Sorin Hermon. «Expressing Reliability with CIDOC CRM». In: *Int. J. Digit. Libr.* 18.4 (2017), pp. 281–287. ISSN: 1432-5012. DOI: 10.1007/s00799-016-0195-1. URL: https://doi.org/10.1007/s00799-016-0195-1.

[NR06] Natasha Noy e Alan Rector. *Defining N-ary Relations on the Semantic Web*. W3C Note. https://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/. W3C, apr. 2006.

[OGO21] Fabrizio Orlandi, Damien Graux e Declan O'Sullivan. «Benchmarking RDF Metadata Representations: Reification, Singleton Property and RDF». In: *15th IEEE International Conference on Semantic Computing, ICSC 2021, Laguna Hills, CA, USA, January 27-29, 2021*. IEEE, 2021. DOI: 10.1109/ICSC50631.2021.00049. URL: https://doi.org/10.1109/ICSC50631.2021.00049.

[Pat14] Peter F. Patel-Schneider Patrick J. Hayes. *RDF 1.1 Semantics*. 2014. URL: https://www.w3.org/TR/rdf11-mt/.

[PRS18] Beatriz Pérez, Julio Rubio e Carlos Sáenz-Adán. «A systematic review of provenance systems». In: *Knowl. Inf. Syst.* 57.3 (2018), pp. 495–543.

[SP20] Leslie F. Sikos e Dean Philp. «Provenance-Aware Knowledge Representation: A Survey of Data Models and Contextualized Knowledge Graphs». In: *Data Sci. Eng.* 5.3 (2020), pp. 293–316. DOI: 10.1007/s41019-020-00118-0. URL: https://doi.org/10.1007/s41019-020-00118-0.

[Zim14] Antoine Zimmermann. *RDF 1.1: On Semantics of RDF Datasets*. Feb. 2014. URL: https://www.w3.org/TR/rdf11-datasets/.

.