Alma Mater Studiorum – Università Di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Inferenze Situate:
Un'implementazione Per GraphDB

Relatore:

Chiar.mo Prof. FABIO VITALI

Presentata da:

ANDREA CORRADETTI

Sessione II - Primo appello

Anno Accademico 2023/2024

Indice

1 - Introduzione

Nell'ambito del Web Semantico, è interessante rappresentare dataset complessi che vanno oltre le semplici asserzioni e che affermano fatti riguardo altre asserzioni.
Queste asserzioni di secondo livello potrebbero esprimere condizioni sulla validità, informazioni su provenienza, autorialità, e livelli di certezza.
Prendiamo come esempio le seguenti proposizioni:

A. Superman can fly
B. LoisLane believes (A)

Diversi approcci e sintassi concrete permettono di rappresentare tali dataset complessi (reificazione standard, rdf-star, e l'uso di *named graphs*).
Per mantenere un linguaggio indipendente dal tipo di rappresentazione, ci riferiremo a questi dataset come "situazionali". Le affermazioni che si riferiscono ad altre affermazioni, come nel caso di B verso A, saranno chiamate "affermazioni situanti"; quelle a cui ci si riferisce, "affermazioni situate". In questa dissertazione vengono usati intercambiabilmente i termini affermazioni, asserzioni e proposizioni.

Delle criticità emergono quando si tenta di realizzare inferenze su dataset situazionali. Per prima cosa, diverse rappresentazioni dei dati e diversi modelli di ragionamento potrebbero necessitare di considerazioni implementative differenti.
Al momento, non esiste un approccio ben definito per le inferenze su dataset situazionali.
I vari modelli di ragionamento attuale non effettuano inferenze su reificazione standard e rdf-star.
I named graphs non possiedono una semantica ben definita e varie implementazioni si comportano in maniera non uniforme nel loro utilizzo.
In generale, non possiamo aspettarci un comportamento uniforme da parte dei triple-store per la gestione delle inferenze.

In secondo luogo, è necessario tenere conto di alcuni comportamenti particolari delle affermazioni situate e situanti in generale.
Il paradosso di Superman mostra come, nel caso di asserzioni riportate, sia possibile partire da premesse vere e ottenere inferenze dalla dubbia validità.

Una formulazione tipica del paradosso di Superman è la seguente [Forbes]:

a. Superman is identical to Clark Kent.
b. Lois Lane believes that Superman can fly.
c. Therefore, Lois Lane believes that Clark Kent can fly.

Anche se (a) e (b) sono vere, resta difficile accettare la proposizione c.

Per evitare situazioni analoghe, i diversi approcci di rappresentazione limitano le inferenze su asserzioni situate.
Il problema è individuabile nell'applicazione su affermazioni riportate del principio di sostituzione, o legge di Leibniz [For96], ed è descritto in [Nel23]. Per evitare inferenze problematiche, alcuni approcci di rappresentazione sono pensati come referenzialmente opachi, ovvero non permettono la sostituzione di riferimenti equivalenti in affermazioni situate. [RDF-STAR23]

Il problema del paradosso di Superman può essere ricondotto ad un caso più generale di inferenze che si basano su premesse valide in senso globale, ma che non sono valide in una specifica località. Nell'esempio, la località dei fatti creduti da Lois Lane non include "Superman is identical to Clark Kent" e la tripla non dovrebbe essere usata per l'inferenza, anche se questa è vera in generale.
Crediamo che, per risultare valide, le inferenze situate debbano utilizzare esclusivamente le regole e le premesse che sono esplicitamente parte della località in questione.

Notiamo anche che una proposizione come "Lois Lane believes that Superman can fly" non afferma alcuna proprietà delle entità menzionate nelle proposizioni riportate.
Pensiamo quindi che eventuali conclusioni raggiunte in una località non ci permettano di derivare proprietà su entità al di fuori della località.

La proposizione in questione è un'ascrizione di uno stato mentale e modella una località che condiziona i valori di verità delle proposizioni contenute. Nello specifico, la località dei fatti creduti da Lois Lane.

In generale, i dataset situazionali si prestano a modellare punti di vista contrastanti o differenze di verità delle proposizioni dovute, ad esempio, a diverse località temporali o spaziali.

Basandoci sulle considerazioni appena introdotte, abbiamo implementato il Situated-Inferences-Addon (SIA), un'estensione per il triple-store GraphDB, con l'obiettivo di realizzare inferenze su dataset situazionali espressi in RDF.
Questa dissertazione discute l'implementazione di SIA, che rappresenta una soluzione al problema delle inferenze situate compatibile con 3 metodi di rappresentazione piuttosto diffusi: rdf-star, reificazione standard, named graphs quotati.
L'estensione ci permette di osservare come, mantenendo isolati i contesi, non si verifichino inferenze problematiche analoghe al paradosso di Superman e riconducibili all'uso implicito di regole e antecedenti che sono al di fuori di una certa località.
Dal momento in cui consideriamo i contesti come località distinte che determinano i valori di verità, diventa possibile utilizzarli per esprimere punti di vista diversi e in disaccordo. SIA è in grado di verificare quali contesti sono internamente consistenti e quali sono in disaccordo tra loro.

Nei suoi processi di inferenza, GraphDB non tiene conto del contesto degli antecedenti e materializza tutte le triple inferite nel default graph; di conseguenza, non possiamo utilizzarlo senza modifiche. [GDB-REASONING]
SIA mette a disposizione alcuni IRI speciali per definire, attraverso query SPARQL, quali grafi di un dataset rdf debbano essere considerati come *contesti situati* e quali come *conoscenza condivisa*. I contesti così dichiarati verranno "presi in carico" dall'estensione, che effettuerà inferenze nei contesti situati utilizzando le triple locali e quelle condivise espressamente. SIA risponderà a successive interrogazioni includendo le asserzioni inferite.
Siccome effettuare inferenze su triple reificate e rdf-star resta una questione complessa, SIA fornisce dell'utilità per convertire affermazioni espresse in questi formati. Le proposizioni risultanti dalla conversione saranno inserite in appositi named graphs, sui quali è possibile svolgere le inferenze.

L'efficacia della versione attuale di SIA risente fortemente di alcune difficoltà incontrate nella fase di sviluppo e, al momento, riusciamo ad implementare correttamente solo una parte di tutte le funzionalità desiderate.

Comunque, riteniamo che sia un primo tentativo promettente di realizzare concretamente inferenze situate appoggiandosi a triple-store commerciali già esistenti.
SIA effettua correttamente inferenze locali nei contesti e, come atteso, osserviamo che non si verificano istanze di paradossi. Siamo anche in grado di determinare correttamente la presenza di contesti in disaccordo.
In ogni caso, non ci sentiamo ancora di consigliare l'utilizzo di SIA fino a che alcune evidenti problematiche non saranno risolte.
In primo luogo, non siamo in grado di generare correttamente tutte le affermazioni *situanti*. Una volta effettuate le inferenze locali, è importante che queste siano situate dalle stesse proposizioni che situano gli antecedenti; sono proprio le preposizioni situanti a determinare una località per le inferenze, ed è necessario restituire i risultati delle inferenze in modo che siano chiaramente collocati con, e qualificati come, i loro antecedenti.

L'attuale implementazione interviene nella valutazione di quasi tutti i pattern di una query; di fatto, vogliamo quasi sempre modificare il risultato restituito includendo le triple inferite. Inoltre, l'ordine dei pattern nella query influenza significativamente il funzionamento dell'estensione. Questi due fattori, insieme ad una sintassi contorta e non conforme al normale utilizzo di SPARQL QUERY, portano ad una soluzione fragile e in parte da ripensare.
Intendiamo correggere queste problematiche delineando più chiaramente le responsabilità di SIA e appoggiandoci maggiormente alle funzionalità di SPARQL QUERY e SPARQL UPDATE.

2 - Premesse del ragionamento su dataset situazionali

Il Contesto tecnologico

In questa sezione introduciamo alcune delle tecnologie principali che rientrano nell'ambito del problema e discutiamo le basi concettuali della nostra soluzione.

Il Web Semantico

Il Web Semantico si propone di estendere i principi del Web dai documenti ai dati.
I dati diventano accessibili utilizzando l'architettura generale del Web, ad esempio attraverso URI, e diventano correlati proprio come lo sono i documenti.
Il Web Semantico definisce e descrive relazioni tra i dati sul web. Il concetto è analogo a quello degli hyperlink nel web corrente, che definiscono relazioni tra la pagina attuale e la pagina di destinazione.
La differenza sostanziale sta nel fatto che le relazioni del web semantico sono tra risorse arbitrarie, non tengono conto di una "risorsa attuale", e sono nominate.

Le seguenti tecnologie sono nate per permettere lo scambio e la manipolazione dei dati del Web Semantico:
RDF è un formato che permette di definire relazioni tra le risorse.
SPARQL è un linguaggio di interrogazione per dati e relazioni.
RDF Schemas e OWL estendono RDF, permettendo di caratterizzare in maniera più granulare i dati e le loro relazioni. [SW-FAQ]

RDF

Il modello RDF permette di dichiarare fatti sotto forma di triple "soggetto-predicato-oggetto", ad esempio "Impiegato38 si chiama Smith".
Il predicato di una tripla è una proprietà specificata con un IRI (una versione internazionalizzata di un URI). Il soggetto e l'oggetto di una tripla possono essere entrambi IRI che fanno riferimento a una qualsiasi entità. L'oggetto può anche essere un valore letterale. [RDF-STAR23]

Semantica di RDF 1.1, in breve

Il significato, in senso ampio, di una affermazione RDF può dipendere da molti fattori, come convenzioni sociali, commenti in lingua naturale, o collegamenti ad altri documenti.
La semantica formale di RDF è espressa mediante teoria dei modelli, cerca di essere neutrale dal punto di vista metafisico e ontologico, e non descrive comprensivamente il significato di una affermazione RDF.
Una semantica formale per RDF permette di determinare quando i processi di inferenza sono validi, cioè quando preservano la verità.
Per una descrizione formale ed esaustiva si rimanda a [RDF-MT04] e [Pat14].

OWL

Il *Web Ontology Language* (OWL) è un linguaggio formale, parte dello *stack* del Web Semantico, pensato per esprimere fatti complessi su entità, gruppi di entità e relazioni.
OWL è un'estensione semantica di RDF. Descrive regole logiche per verificare la consistenza di affermazioni e per derivare nuova conoscenza da fatti espliciti attraverso processi di inferenza.
I documenti OWL sono detti ontologie, e possono essere in relazione con altre ontologie. [OWL]

Tra i vari frammenti di OWL, citiamo OWL 2 RL che mantiene una buona espressività e tempi di inferenze ragionevoli nel contesto della soluzione che andremo a discutere.
Vedremo che l'utilizzo di OWL su dataset situazionali porta ad inferenze non corrette e discuteremo alcune possibili soluzioni.

Triple come risorse

A volte, vogliamo che il soggetto o l'oggetto di una tripla facciano riferimento ad un'altra tripla, come nel caso di "A detta di Lois Lane, Superman sa volare". Questa asserzione può essere modellata con "Superman sa volare" come soggetto, "a detta di" come predicato, e "Lois Lane" come oggetto.
Esistono vari formati per esprimere asserzioni situate come le precedenti. Nell'ambito del Web Semantico vengono regolarmente usati reificazione standard rdf, rdf-star e quoted named graphs.
Alcune problematiche emergono quando si tenta di svolgere inferenze su affermazioni situate espresse in questi formati.
Al di là delle differenze di rappresentazione, sarà necessario osservare alcune peculiarità delle affermazioni situate in generale per evitare che le inferenze introducano risultati indesiderati.

Di seguito, introduciamo il paradosso di Superman, un esempio tipico in letteratura quando si tratta di inferenze su dataset situazionali. Parleremo anche del concetto di opacità referenziale, introdotto per evitare il paradosso in questione e altre problematiche affini.

Basi concettuali del ragionamento su dataset Situazionali

Opacità referenziale e il paradosso di Superman

Il termine opacità referenziale indica l'impossibilità di sostituire riferimenti equivalenti ma sintatticamente differenti all'interno di affermazioni che ascrivono stati mentali e, come vedremo, affermazioni situate in generale.
Non osservare questa peculiarità delle affermazioni situate porta all'introduzione di significati nuovi e indesiderati, e di inferenze dalla dubbia validità

Il concetto è esemplificato dal paradosso di Superman [Forbes]:

a. Superman is identical to Clark Kent.
b. Lois Lane believes that Superman can fly.
c. Therefore, Lois Lane believes that Clark Kent can fly.

In questo caso, la regola di inferenza utilizzata è il principio di indiscernibilità degli identici conosciuto anche come legge di Leibniz, o principio di sostituzione, che afferma che "se due oggetti sono uguali, allora godono delle stesse proprietà" [For96]. Come tali, i due oggetti dovrebbero essere sostituibili in una generica proposizione senza alterarne il valore di verità.
Sia (a) che (b) sono proposizioni vere, ma l'affermazione (c), risultante dall'inferenza, risulta difficile da accettare.

Il problema sembra sorgere nel caso di asserzioni indirette, citate, o riportate, che non affermano fatti o proprietà delle entità menzionate in assoluto, ma fatti o proprietà il cui valore di verità è circoscritto ad una certa località delimitata dalle affermazioni situanti.
Rientrano tra queste le asserzioni che ascrivono stati psicologici ad un attore, come potrebbe essere "Lois Lane believes X", o asserzioni che definiscono località temporali o spaziali.

Applicare la legge di Leibniz sulla proposizione b sembra apparentemente lecito in quanto la proposizione riportata sembra dichiarare una proprietà dell'entità Superman e, se presa in isolamento, risulta vera.

Facciamo però notare che, riportando un'affermazione, stiamo affermando qualcosa riguardo l'agente e l'affermazione riportata, ma non riguardo il *contenuto* dell'affermazione riportata. Ad esempio, "Lois Lane believes that Superman can fly" afferma qualcosa riguardo Lois Lane e riguardo la frase "Superman can fly", ma non possiamo dedurre nulla riguardo l'entità Superman partendo da quest'ultima.

Il problema del paradosso di Superman e istanze simili di inferenze problematiche si basano su due forti assunzioni: che il principio di sostituzione sia applicabile in generale, e in particolare nel caso di affermazioni situate; e che le regole di inferenza che giustificano tale sostituzione siano applicabili dappertutto.

È facile generare controesempi che mostrano come l'applicazione indiscriminata del principio di sostituzione su affermazioni situate permetta di cambiare il valore di verità di una proposizione (se considerata in assoluto) o di generare proposizioni insensate.
Ad esempio: Partendo dalla proposizione vera "Lois Lane thinks that Superman is not Clark Kent", potremmo generare "Lois Lane thinks that Clark Kent is not Clark Kent" che è una contraddizione.

Riteniamo ragionevole sostenere che, nel caso generale, le proposizioni riportate non ammettono l'applicazione della regola di sostituzione, in quanto non asseriscono proprietà delle entità menzionate.

Istanza di un problema più esteso

Nell'ambito del Web semantico, il paradosso di Superman e la questione dell'opacità referenziale sono stati discussi con l'introduzione della reificazione rdf e relativamente al discorso sulle proposizioni citate/riportate.
[Pru12], in una delle prime discussioni sul tema, usa il paradosso di Superman come critica alla reificazione RDF e come motivazione per bloccarne o limitarne l'uso nelle inferenze. Lo stesso problema si ripresenta successivamente in maniera analoga con l'introduzione delle *quoted triple* [RDF-STAR23] e dei *quoted graphs* [N3].

Una rappresentazione del paradosso di superman espressa in rdf-star è la seguente:

```
#### under OWL-entailment
:superman owl:sameAs :clark.
<< :superman :can :fly >> :reportedBy :loislane.
#### does NOT entail
<< :clark :can :fly >> :reportedBy :loislane.
```

Rdf-star è detto *referenzialmente opaco* in quanto riferimenti equivalenti ma sintatticamente differenti non possono essere sostituiti nel caso di affermazioni riportate. In Rdr-Star, il principio di sostituzione viene bloccato espressamente. [RDF-STAR23]

In [Pru12], viene però notato che non tutte le sostituzioni sono problematiche. Ad esempio, sostituire Superman con Man of Steel risulta sensato e potrebbe essere desiderabile.
Guardando con più attenzione, ci rendiamo conto che la sostituzione di dubbia correttezza non è una caratteristica implicita di RDF, ma è la conseguenza di una regola di inferenza giustificata dal predicato owl:SameAs. Il problema nasce dall'inclusione di OWL, che non è pensato per funzionare su strutture più elaborate delle semplici triple. Non potendo aspettarci inferenze corrette su affermazioni riportate, non dovremmo utilizzarlo nel suo stato attuale su queste ultime.

Riteniamo che il paradosso di Superman e altre problematiche di natura simile, siano istanze di inferenze locali basate su alcune proposizioni vere in generale ma che non dovrebbero entrare a far parte della località. Nell'esempio precedente Lois Lane **non** crede che Superman e Clark Kent siano la stessa persona e, come tale, la proposizione non dovrebbe essere utilizzata per l'inferenza.

Assunzioni

A questo punto ci chiediamo quali inferenze sia lecito accettare in un contesto situato e quali tra queste possano essere usate per trarre conclusioni che esulano dalla località del contesto.
Viste le considerazioni precedenti, riteniamo plausibile considerare che proposizioni e regole di inferenza valide in generale non siano implicitamente trasferibili all'interno delle località create dalle affermazioni situanti.

Le premesse e le regole di inferenza devono essere tutte parte della località.
Inoltre, siccome le affermazioni situate non descrivono nulla al di fuori della località, non possiamo utilizzare le inferenze per trarre conclusioni sulle entità menzionate al di fuori del contesto situato.

Nella nostra soluzione, terremo conto delle seguenti assunzioni basate sulla discussione precedente:

1. Razionalità: le inferenze all'interno di un contesto situato devono funzionare come all'esterno. Ci aspettiamo dunque che la regola di modus ponens valga all'interno delle situazioni, sia che esse rappresentino stati mentali di un agente o delimitazioni di carattere generale, ad esempio temporale o spaziale.
2. Permeazione: Tutte le premesse devono essere parte del contesto situato. Nel caso di agenti, esse devono essere conosciute ed accettate soggettivamente.
3. Località: Non possiamo trarre conclusioni in generale da inferenze situate. Le conclusioni restano nella località del contesto.

L'assunzione 1 è necessaria per far sì che sia possibile effettuare inferenze.
Le assunzioni 2 e 3 circoscrivono le inferenze situate ad un particolare contesto, isolandole dall'esterno.

Cosa intendiamo fare

Nei capitoli che seguono, andiamo a descrivere la realizzazione di Situated-Inferences-Addon (SIA), un'estensione per il Triple-Store GraphDB che vuole effettuare inferenze tenendo conto delle assunzioni appena delineate.
Vorremmo che le inferenze siano circoscritte ad una certa località (un contesto situato) e che vi sia la possibilità di dichiarare espressamente un certo grado di permeazione delle triple da fuori i contesti a dentro i contesti (i contesti situati potranno attingere ad una conoscenza condivisa).
Desideriamo che le triple inferite siano situate dalle stesse proposizioni che situano gli antecedenti.
Vorremmo che le località siano in grado di esprimere informazioni in disaccordo, in modo da poter modellare, ad esempio, diversi punti di vista, differenze di verità in periodi temporali differenti, o diversi livelli di certezza. Vorremmo inoltre essere in grado di individuare questi disaccordi.

In tutto questo, ci aspettiamo che non si verifichino istanze di inferenze problematiche riconducibili al paradosso di superman

Problematiche simili e approcci preesistenti

Alcune discussioni portate avanti negli anni sono rilevanti alla soluzione trattata da questa dissertazione.
Context OWL, discusso in [BGvH+03], è una proposta di estensione di OWL che ha lo scopo di riconciliare ontologie separate e internamente consistenti. Le ontologie possono essere considerate come contesti separati i cui concetti vengono mappati e messi in relazione da alcune regole.
Il nostro approccio non intende riconciliare ontologie differenti, ma vuole permettere la rappresentazione nella stessa ontologia di località (generalmente punti di vista) che possono essere in disaccordo tra loro.

In [FFP+09], gli autori intendono gestire informazioni di provenienza di triple RDF implicite (le triple inferite e non asserite). Propongono l'utilizzo di un "colore" (un quarto IRI per ogni tripla, analogo al concetto di grafo) e un modello di inferenza per le triple colorate in modo da determinare la provenienza delle triple inferite.
Il colore della tripla inferita sarà il risultato di un operatore applicato ai colori delle triple antecedenti.
Il concetto è comparabile al nostro approccio ai contesti situati. Infatti, antecedenti con lo stesso colore (stesso contesto) generano triple con lo stesso colore (stesso conteso)
Nell'articolo, gli autori non definiscono una semantica per i colori, ma riconoscono la possibilità di effettuare inferenze sulle triple colorate e riconoscono che la provenienza delle triple può ricoprire un ruolo importante nel processo di inferenza.
Nel nostro caso, la semantica del contesto è piuttosto intuitiva: il risultato dell'inferenza appartiene alla stessa situazione, ovvero è situato dalle stesse triple che situano gli antecedenti.

In [NS17], gli autori tentano di formalizzare un approccio alle inferenze per triple contestualizzate utilizzando la semantica Singleton Property descritta in [NV14].
Il nostro approccio invece è descritto facilmente dalla semantica 3.4 dei named graphs, descritta in [RDF-DS14]

In [ABF19], per gestire il ragionamento su triple contestualizzate, gli autori propongono un'estensione di OWL chiamata OWL-C, in cui le classi, le proprietà e gli assiomi possono essere contestuali o non-contestuali. Sono definiti anche operatori per permettere al meta-livello e al livello-oggetto di interagire. Le regole definite permettono interazioni complesse solo tra proposizioni nello stesso grafo o tra un grafo e il default graph.
OWL-C si aspetta anche che tutte le triple owl siano collocate nel default graph. C'è dunque una chiara differenza tra T-Box (nel default graph) e A-Box (nel default graph o un altro contesto)
Nel nostro approccio, il default graph non ha alcuna semantica particolare e può essere utilizzato come conoscenza condivisa al pari di un qualsiasi altro grafo. Le asserzioni TBox e ABox possono essere distribuite tra conoscenza condivisa e grafi privati, con conseguente differenza di visibilità.

3 - SIA, un'implementazione per il ragionamento su dataset situazionali

Situated-Inferences Add-on (SIA) è un estensione del triple-store GraphDB[1] ed è pensata per consentire il ragionamento su dataset situazionali che utilizzano named graphs, quoted triples o reificazione standard.
I tre formati scelti sono frequentemente usati per modellare affermazioni riportate/situate. Nonostante le differenze degli approcci, SIA dimostra la possibilità di svolgere inferenze su tutti e tre.

SIA fornisce una sintassi per dichiarare espressamente quali grafi debbano essere trattati come grafi situati, sui quali vengono effettuate inferenze circoscritte al singolo contesto. SIA permette di generare named graphs partendo da triple situate espresse con RDF-Star e reificazione standard, di effettuare inferenze su questi named graphs, e di riconvertire il risultato nel formato desiderato. Offre anche un predicato speciale per stabilire quali contesti esprimono informazioni in disaccordo.

L'estensione è reperibile attraverso il link nella sezione allegati e a piè di pagina.[2]
Di seguito mostriamo come utilizzare SIA per interagire con i vari formati.

Nota tecnica:

Gli esempi sono da intendersi in sintassi concreta Trig-Star. Per una definizione formale, consultare [TRIG14] e [RDF-STAR23]
Tutti gli esempi che seguono usano i seguenti prefissi, a meno che non siano preceduti da indicazioni diverse.

```
PREFIX :          <http://a#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:      <http://www.w3.org/2000/01/rdf-schema#>
PREFIX conj:      <https://w3id.org/conjectures/>
```

[1] https://graphdb.ontotext.com/
[2] https://github.com/andrea-corradetti/situated-inferences

Supponiamo di avere sempre a disposizione il seguente dataset di partenza:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX conj: <https://w3id.org/conjectures/>
PREFIX : <http://a#>
PREFIX t: <http://t#>

INSERT DATA {
	graph conj:sharedKnowledge {
		:Superman a t:Person.
		:ClarkKent a t:Person.

		:LoisLane a t:Person.
		:MarthaKent a t:Person.
		:I a t:Person.

		t:FictionalPerson rdfs:subClassOf t:Person.
		t:RealPerson rdfs:subClassOf t:Person.
		t:FictionalPerson owl:complementOf t:RealPerson.

		:fly a t:flyingPower.
		:clingFromCeiling a t:spiderLikePower.

		t:flyingPower rdfs:subClassOf t:supernaturalPower.
		t:spiderLikePower rdfs:subClassOf t:supernaturalPower.

		t:SuperHero rdfs:subClassOf t:Person;
			owl:onProperty :can;
			owl:someValuesFrom t:supernaturalPower.
		t:FlyingSuperHero rdfs:subClassOf t:SuperHero;
				owl:onProperty :can;
				owl:someValuesFrom t:flyingPower.
		t:SpiderSuperHero rdfs:subClassOf t:SuperHero;
```

```
                owl:onProperty :can;
                owl:someValuesFrom t:spiderLikePower.
        t:FlyingSuperHero owl:disjointWith t:SpiderSuperHero .
    }
}
```

Inferenze Locali su Named Graphs Situati

Ogni named graph è una coppia composta da un IRI o un nodo vuoto (il nome del grafo) e un grafo RDF. I nomi dei grafi sono unici all'interno di un dataset RDF.
Il nome del grafo non denota necessariamente il grafo stesso ed è solamente accoppiato sintatticamente al grafo. RDF non impone alcuna restrizione formale su quale risorsa il nome del grafo possa denotare. [RDF-C14]

In [RDF-DS14] viene introdotto il concetto di dataset rdf e 8 semantiche possibili vengono proposte per i named graphs.
Nessuna viene preferita alle altre per evitare di contraddire implementazioni già esistenti.
La semantica 3.4 descrive i named graphs come contesti a cui sono circoscritti i valori di verità delle triple contenute. Questa semantica potrebbe modellare l'evoluzione di fatti nel tempo (il contesto definisce un lasso di tempo in cui le affermazioni sono vere) o potrebbe esprimere opinioni contrastanti (ogni contesto rappresenta le opinioni di un agente). Questa semantica si adatta bene al nostro obiettivo di realizzare inferenze situate, ma non possiamo applicarla indiscriminatamente ad ogni grafo.

SIA utilizza due classi, `conj:situatedContext` e `conj:sharedKnowledge`, per dichiarare espressamente quali grafi debbano essere considerati come contesti situati e quali debbano collaborare nel calcolo delle inferenze locali. Fa successivamente uso dei predicati `conj:situateSchema` e `conj:situate` per permettere all'utente di richiedere le inferenze.
Al momento dell'interrogazione, il plugin calcola la chiusura dei contesti situati attingendo alle asserzioni assiomatiche[3], quelle locali del contesto stesso, e quelle di ogni contesto di *shared knowledge*.

[3] Sono tutte le triple asserite esplicitamente dal modello di inferenze selezionato, e.g. RDFS, OWL 2 RL

Le asserzioni inferite saranno restituite come parte del contesto dalle query SPARQL e saranno correttamente indicate come implicite.
Le triple inferite non saranno materializzate in automatico. L'utente può successivamente effettuare un inserimento esplicito utilizzando una normale operazione di update SPARQL con pattern matching.
Ne mostriamo il funzionamento con il seguente esempio.

Es 3.1 Inferenze locali su contesti situati

Inseriamo queste triple aggiuntive che definiscono 3 località. Ci aspettiamo inferenze diverse nei 3 contesti.

```
INSERT DATA {
	:LoisLane :thinks :LoisLanesThoughts
	GRAPH :LoisLanesThoughts {
		:Superman :can :fly .
		:Superman owl:differentFrom :ClarkKent.
		:Superman a t:RealPerson .
	}
	:MarthaKent :thinks :MarthaKentsThoughts
	GRAPH :MarthaKentsThoughts {
		:Superman :can :fly .
		:Superman owl:sameAs :ClarkKent.
		:Superman a t:RealPerson .
	}

	:I :thinks :myThoughts
	GRAPH :myThoughts {
		:Superman :can :clingFromCeiling .
		:Superman owl:sameAs :ClarkKent.
		:Superman a t:FictionalPerson .
	}
}
```

Vorremmo che in ognuno di questi grafi vengano svolte inferenze utilizzando le triple nel grafo `conj:sharedKnowledge`. Utilizziamo la seguente query:

```
PREFIX conj: <https://w3id.org/conjectures/>
PREFIX rdf4j: <http://rdf4j.org/schema/rdf4j#>
PREFIX schemas: <https://w3id.org/conjectures/schemas/>
PREFIX : <http://a#>

select ?s ?p ?o ?g1
from conj:situations  #comunica al plugin di intervenire. Non è un vero grafo
where {
	#?task è un oggetto di lavoro che mantiene informazioni sull'inferenza
	?task conj:situateSchema schemas:thoughts.

	graph schemas:thoughts {
		#questo grafo è condiviso
		conj:sharedKnowledge a conj:SharedKnowledgeContext.

		#i  grafi che seguono sono isolati
		:LoisLanesThoughts a conj:SituatedContext.
		:MarthaKentsThoughts a conj:SituatedContext.
		:myThoughts a conj:SituatedContext.

		#vogliamo che i grafi situati siano rinominati
		"-situated" a conj:appendToContexts.
	}

	#tutti i grafi situati verranno legati a ?g1
	?task conj:hasSituatedContext ?g1.

	#cerchiamo tutte le triple nei grafi situati
	graph ?g1 {
		?s ?p ?o .
	}
}
```

Riportiamo alcune porzioni di risultati interessanti.

	s	p	o	g1
1	:Superman	rdf:type	t:Person	:LoisLanesThoughts-situated
2	:Superman	rdf:type	t:RealPerson	:LoisLanesThoughts-situated
3	:Superman	rdf:type	t:FlyingSuperHero	:LoisLanesThoughts-situated
4	:Superman	rdf:type	t:SuperHero	:LoisLanesThoughts-situated
5	:Superman	rdf:type	t:Person	:MarthaKentsThoughts-situated
6	:Superman	rdf:type	t:RealPerson	:MarthaKentsThoughts-situated
7	:Superman	rdf:type	t:FlyingSuperHero	:MarthaKentsThoughts-situated
8	:Superman	rdf:type	t:SuperHero	:MarthaKentsThoughts-situated
9	:Superman	rdf:type	t:Person	:myThoughts-situated
10	:Superman	rdf:type	t:FictionalPerson	:myThoughts-situated
11	:Superman	rdf:type	t:SpiderSuperHero	:myThoughts-situated
12	:Superman	rdf:type	t:SuperHero	:myThoughts-situated

Notiamo che in tutti e tre i contesti è comparsa la tripla `:Superman rdf:type t:Person.`
In `:LoisLanesThoughts e :MarthaKentsThoughts`, l'inferenza è dovuta a `:Superman a t:RealPerson` e `t:RealPerson rdfs:subClassOf t:Person`.

In `:myThoughtsSituated`, abbiamo `:Superman a t:FictionalPerson`, che insieme a `t:FictionalPerson rdfs:subClassOf t:Person` produce il risultato osservato.
In `:myThoughtsSituated`, compare `:Superman rdf:type t:SpiderSuperHero` perché è inclusa la tripla `:Superman :can :clingFromCeiling.`

44	:Superman	rdf:type	t:FictionalPerson	:myThoughts-situated
45	:Superman	:can	:clingFromCeiling	:myThoughts-situated
46	:Superman	rdf:type	t:SpiderSuperHero	:myThoughts-situated
47	:Superman	rdf:type	t:SuperHero	:myThoughts-situated

Negli altri due grafi, invece, abbiamo i seguenti risultati dovuti alla tripla `:Superman :can :fly`.

	s	p	o	g1
1	:Superman	rdf:type	t:FlyingSuperHero	:LoisLanesThoughts-situated
2	:Superman	rdf:type	t:FlyingSuperHero	:MarthaKentsThoughts-situated

Notiamo anche che nel risultato non ci sono istanze di permeazione indesiderate di `owl:sameAs`. Difatti, `:LoisLane` non deduce che `:ClarkKent` è un `t:SuperHero`.

clarkkent type super

Showing results from 1 to 319 of 319. Query took 0.2s, today at 22:11.

	s	p	o	g1
1	http://a#ClarkKent	rdf:type	http://t#FlyingSuperHero	http://a#MarthaKentsThought situated
2	http://a#ClarkKent	rdf:type	http://t#SuperHero	http://a#MarthaKentsThought situated
3	http://a#ClarkKent	rdf:type	http://t#SpiderSuperHero	http://a#myThoughts-situated
4	http://a#ClarkKent	rdf:type	http://t#SuperHero	http://a#myThoughts-situated

Es 3.2 Inserire le triple inferite

Il seguente update permette di inserire le triple inferite in `:LoisLanesThoughts`. Facciamo notare che le triple saranno marcate come esplicite da GraphDB.

```
PREFIX conj: <https://w3id.org/conjectures/>
PREFIX rdf4j: <http://rdf4j.org/schema/rdf4j#>
PREFIX schemas: <https://w3id.org/conjectures/schemas/>
PREFIX : <http://a#>

insert {
	graph ?g1 {
		?s ?p ?o
	}
}
using conj:situations #analogo di FROM per gli insert
where {
	?task conj:situateSchema schemas:thoughts.

	graph schemas:thoughts {
		rdf4j:nil a conj:SharedKnowledgeContext.
		:LoisLanesThoughts a conj:SituatedContext.
		:MarthaKentsThoughts a conj:SituatedContext.
		:myThoughts a conj:SituatedContext.
	}

	bind (:LoisLanesThoughts as ?g1).

	graph ?g1 {
		?s ?p ?o .
	}
}
```

Inferenze su Reificazione Standard

RDF definisce un vocabolario specifico per descrivere triple RDF.
Consideriamo la tripla

```
:Superman :can :fly .
```

Supponiamo di avere un IRI `:S` che possa essere usato per riferirsi alla tripla.
Allora il seguente grafo descrive il precedente:

```
:S rdf:type rdf:Statement .
:S rdf:subject :Superman .
:S rdf:predicate :can .
:S rdf:object :fly .
```

Il secondo grafo è chiamato una "reificazione" della tripla nel primo grafo.
Le triple identificate in questo modo possono essere ulteriormente descritte (diremmo situate).
La tripla che segue asserisce qualcosa riguardo la reificazione (la situa):

```
:S :accordingTo :LoisLane .
```

Una reificazione descrive relazioni tra un *token* di una tripla e alcune risorse.
Il valore di una proprietà è la risorsa denotata dall'IRI e non l'IRI stesso: non stiamo menzionando l'IRI.
La reificazione di una tripla non implica la tripla e non è implicata da essa.
La reificazione descrive e afferma l'esistenza del *token* di una tripla ma non la asserisce. D'altro canto, asserire una tripla non implica asserire che ne esista un *token*/istanza. [Pat14]

I modelli di ragionamento esistenti non offrono inferenze su reificazione standard.
SIA permette di convertire tutte le affermazioni reificate in normali triple. Le triple generate verranno raccolte in named graphs che rappresentano le situazioni.

Ad esempio:

```
:S1 rdf:type rdf:Statement .
:S1 rdf:subject :Superman .
:S1 rdf:predicate :can .
```

```
:S1 rdf:object :fly .
:LoisLane :thinks :S1 .

:S2 rdf:type rdf:Statement .
:S2 rdf:subject :Superman .
:S2 rdf:predicate owl:differentFrom .
:S2 rdf:object :ClarkKent .
:LoisLane :thinks :S2 .
```

Verrà convertito in

```
graph :LoisLaneThoughts {
      :Superman :can :fly .
      :Superman owl:differentFrom :ClarkKent .
}

:LoisLane :thinks :LoisLaneThoughts.
```

Il contesti risultanti da queste operazioni possono essere successivamente presi in carico per inferenze locali.

Es 3.3 Inferenze con Reificazione Standard

Al dataset di partenza aggiungiamo le seguenti triple:

```
REFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX t: <http://t#>
PREFIX : <http://a#>

insert data {
      :S rdf:type rdf:Statement .
      :S rdf:subject :Superman .
      :S rdf:predicate :can .
      :S rdf:object :clingFromCeiling .
```

```
        :I :thinks :S.
        :S :since "2023".
}
```

Vorremmo che `:Superman :can :clingFromCeiling` fosse usato per le inferenze, e vorremmo che tutte le nuove triple T siano situate da `:I :thinks T e T :since "2023".`

Usiamo la seguente query:

```
PREFIX conj: <https://w3id.org/conjectures/>
PREFIX rdf4j: <http://rdf4j.org/schema/rdf4j#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX schemas: <https://w3id.org/conjectures/schemas/>
PREFIX : <http://a#>

select distinct ?subject ?predicate ?object
from conj:situations
where {
        #trasformiamo la reificazione in una tripla in un grafo temporaneo
        :S conj:asSingleton conj:singleton.

        #shorthand, il primo grafo è quello da situare, gli altri sono condivisi
        ?situation conj:situate (conj:singleton rdf4j:nil conj:sharedKnowledge).

        #vogliamo generare tutti gli statements reificati dal grafo temporaneo
        :reifiedGraph conj:reifiesGraph ?situation

        graph :reifiedGraph {
                ?subject ?predicate ?object.
        }
}
```

Otteniamo i seguenti risultati

	s	p	o
1	https://w3id.org/conjectures/reifications/212-1-229	rdf:type	rdf:Statement
2	https://w3id.org/conjectures/reifications/212-1-229	rdf:subject	http://a#Superman
3	https://w3id.org/conjectures/reifications/212-1-229	rdf:predicate	rdf:type
4	https://w3id.org/conjectures/reifications/212-1-229	rdf:object	http://t#SpiderSuperHero

È stata inferita la tripla `:Superman rdf:type t:SpiderSuperHero` che è stata reificata.
Ricordiamo che siamo partiti da queste triple:

```
:S rdf:type rdf:Statement .
:S rdf:subject :Superman .
:S rdf:predicate :can .
:S rdf:object :clingFromCeiling .
```

Nella versione attuale, le triple situanti non sono generate correttamente.

Inferenze su RDF-Star

Rdf-star rappresenta un'alternativa compatta alla reificazione standard.
La seguente è una tripla che utilizza la sintassi concreta Turtle-star.

```
<< :Superman :can :fly >> :accordingTo :LoisLane .
```

Rdf-star estende il modello RDF con un nuovo costrutto: le *quoted triples* (triple citate).
Al contrario della reificazione standard, che permette più *token* della stessa tripla, ogni volta che una tripla citata appare, questa denota la stessa cosa.
Le triple citate godono di opacità referenziale: due triple che contengono termini sintatticamente distinti ma equivalenti (nel dettaglio, due identificatori diversi per la stessa entità) sono considerate triple diverse.
Le *quoted triples* non sono asserite.

In maniera analoga a sopra, SIA permette di riunire in un contesto le *quoted triples* situate dalla stessa affermazione.

Ad esempio:

```
:LoisLane :thinks <<:Superman :can :fly>> .
:LoisLane :thinks <<:Superman owl:differentFrom :ClarkKent>> .
```

Verrà convertito in:

```
graph :LoisLaneThoughts {
	:Superman :can :fly .
	:Superman owl:differentFrom :ClarkKent .
}

:LoisLane :thinks :LoisLaneThoughts.
```

Questo contesto, equivalente al precedente può essere utilizzato per inferenze locali.

Es 3.4 Inferenze con Rdf-Star

Aggiungiamo la seguente tripla al dataset di partenza:

```
PREFIX : <http://a#>
PREFIX t: <http://t#>

insert data {
	:I :thinks << :Superman :can :clingFromCeiling >>.
}
```

Eseguiamo la seguente query:

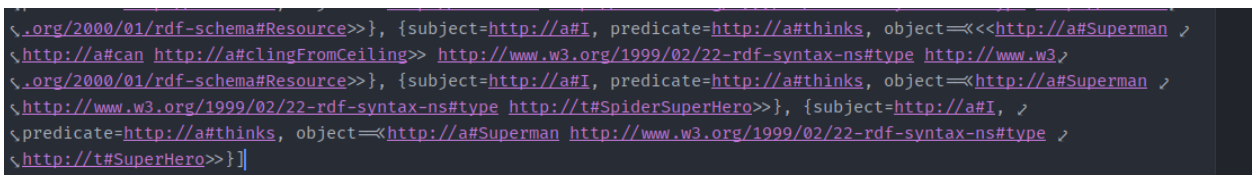
```
PREFIX conj: <https://w3id.org/conjectures/>
PREFIX rdf4j: <http://rdf4j.org/schema/rdf4j#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://a#>
```

```
select distinct ?subject ?predicate ?object ?g from conj:situations where {
	:I :thinks ?triple.
	bind (conj:situations\/I-thinks as ?g)
	?g conj:groupsTriple ?triple.
	conj:task conj:situateSchema conj:schemas\/s.
	graph conj:schemas\/s {
		?g a conj:SituatedContext.
		conj:sharedKnowledge a conj:SharedKnowledgeContext.
	}

	#vogliamo che le triple inferite nel contesto siano "espanse" in rdf-star
	conj:expanded conj:expands ?g

	graph ?g {
		?subject ?predicate ?object
	}
}
```

Il seguente frammento mostra che nel grafo è presente `:Superman rdf:type t:SpiderSuperHero`

	subject	predicate	object	g
1	:Superman	rdf:type	t:Person	conj:situations/I-thinks
2	:Superman	rdf:type	owl:Thing	conj:situations/I-thinks
3	:Superman	rdf:type	t:SpiderSuperHero	conj:situations/I-thinks
4	:Superman	rdf:type	t:SuperHero	conj:situations/I-thinks

Al momento le triple situanti non sono generate correttamente in GraphDB.
Riusciamo a generare il seguente risultato nell'ambiente di test.



Notiamo la tripla `:I :thinks <<:Superman rdf:type t:SpiderSuperHero>>.`

Ricerca dei disaccordi

SIA è in grado di verificare se un singolo contesto è internamente consistente controllando che nessuna regola di consistenza sia violata dalle asserzioni nel contesto in questione.
Se il contesto è situato, SIA terrà conto delle asserzioni nel contesto originario, di quelle assiomatiche e di tutte quelle nella shared knowledge, e ci informerà se qualcuna di queste componenti è inconsistente.
Le singole componenti potrebbero essere consistenti in isolamento ma la loro unione potrebbe essere inconsistente. In tal caso, SIA ci informerà che il contesto situato nell'interezza è inconsistente.

Presi due contesti arbitrari, possiamo verificare la presenza di un eventuale disaccordo.
SIA controlla innanzitutto la consistenza dei singoli contesti, e successivamente la consistenza dell'unione. Se i singoli contesti sono consistenti ma la loro unione non lo è, allora siamo di fronte ad un disaccordo.

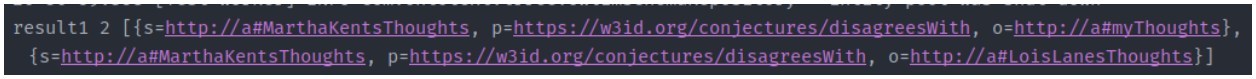
Es 3.5 Ricerca dei disaccordi

Utilizziamo questa query per trovare tutti i contesti in disaccordo con `:MarthaKentsThoughts`:

```
PREFIX conj: <https://w3id.org/conjectures/>
PREFIX rdf4j: <http://rdf4j.org/schema/rdf4j#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://a#>

select distinct ?s ?p ?o from conj:situations where {
	graph conj:schemas\/s {
		:MarthaKentsThoughts a conj:SituatedContext.
		:myThoughts a conj:SituatedContext.
		rdf4j:nil a conj:SharedKnowledgeContext.
	}

	conj:task conj:situateSchema conj:schemas\/s.
```

```
        VALUES (?s ?p) { (:MarthaKentsThoughts conj:disagreesWith) }
        ?s ?p ?o
}
```

Otteniamo il seguente risultato:[4]



Difatto `:MarthaKentsThoughts e :LoisLanesThoughts` sono in disaccordo sul fatto che ClarkKent e Superman siano la stessa entità.
`:myThoughts e :MarthaKentsThoughts` sono in disaccordo sul fatto che superman sia una persona reale o di finzione.

Se `:MarthaKentsThoughts` fosse stato inconsistente, avremmo ottenuto solamente il risultato `:MarthaKentsThoughts conj:disagreesWith :MarthaKentsThoughts.`

Espansione dei contesti situati

Tutte le triple in un contesto situato possono essere rese disponibili alle query SPARQL, sia in forma di asserzione reificata situata che di asserzione RDF-star.

Supponiamo di avere il seguente grafo situato da una qualche tripla. Le prime due triple sono state inserite dall'utente, la terza è il risultato di inferenze locali, la quarta è la tripla situante:

```
graph :LoisLaneThoughts {
        :Superman rdf:type :Superhero .
        :Superhero rdfs:subClassOf :Person .
        :Superman rdf:type :Person .
}

:LoisLane :thinks :LoisLaneThoughts.
```

SIA renderà disponibile tutte le seguenti triple:

[4]Il risultato mostrato è preso dall'ambiente di test

```
:LoisLane :thinks << :Superman rdf:type :Superhero >>.
:LoisLane :thinks << :Superhero rdfs:subClassOf :Person >>.
:LoisLane :thinks << :Superman rdf:type :Person >>.
:LoisLane :thinks conj:S1.
:LoisLane :thinks conj:S2.
:LoisLane :thinks conj:S3.
```

Dove `conj:S1`, `conj:S2`, `conj:S3` sono token descritti da reificazione standard. Ne riportiamo una reificazione per esteso, le altre sono analoghe:

```
conj:S2 rdf:type rdf:Statement .
conj:S2 rdf:subject :Superhero .
conj:S2 rdf:predicate rdfs:subClassOf .
conj:S2 rdf:object :Person .
```

SIA permette dunque di effettuare inferenze su contesti situati, espressi sia con reificazione standard che rdf-star, in maniera conforme alle assunzioni elencate nel capitolo 2. Le inferenze restano locali, senza entrare a far parte della repository, e tengono conto solo di proposizioni che sono espressamente parte della località della situazione.

Differenze con la strategia di inferenza di GraphDB

Normalmente, all'inserimento, GraphDB effettua forward-chaining con materializzazione nel *default graph* delle triple inferite.

La strategia di forward chaining consiste nell'applicare le regole di inferenza ad alcune proposizioni di partenza (nello specifico, le asserzioni inserite esplicitamente e quelle assiomatiche) per generare nuove proposizioni. Le regole vengono iterativamente riapplicate alla combinazione di proposizioni di partenza e proposizioni inferite fino a che si generano nuove proposizioni.

Con "materializzazione" intendiamo che le nuove triple sono calcolate una volta e salvate, per poi essere rese disponibili a tutte le query.[5]

[5] Questo rende i processi di inserimento e di cancellazione relativamente costosi, in quanto diventa necessario rieffettuare le inferenze. Le interrogazioni però sono molto rapide: le triple inferite sono già state calcolate.

Nell'effettuare inferenze, GraphDB non tiene conto in alcun modo del contesto delle triple: tutti gli statement della repository sono utilizzati e le inferenze vengono materializzate nel default graph. [GDB-REASONING]

SIA utilizza la stessa strategia di forward-chaining, ma lo fa al momento di query e tenendo conto solo dei contesti che l'utente ha dichiarato di voler usare. I `conj:situatedContext` verranno tenuti isolati e condivideranno solo le triple nei contesti dichiarati `conj:sharedKnowledge`. Le triple non vengono materializzate.

Come installare SIA

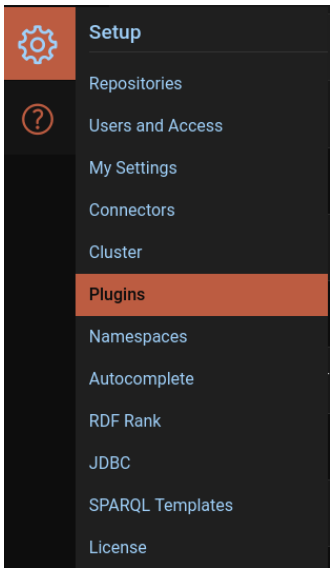
Una volta recuperata una copia della repository, disponibile tramite il link nella sezione allegati, eseguiamo ./gradlew build per effettuare test e assemblaggio degli artefatti.

Se si è interessati a saltare la fase di test, basta usare il comando ./gradlew assemble. L'artefatto risultante va collocato nella cartella $GRAPHDB-HOME/lib/plugins/situated-inferences.

$GRAPHDB-HOME è la cartella di installazione di GraphDB.

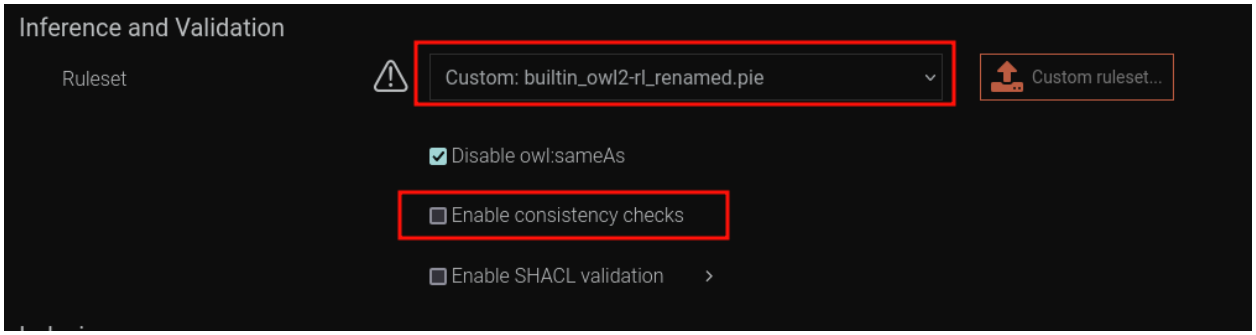
Accortezze di configurazione per SIA

Per prima cosa, è necessario verificare che il plugin sia attivo nella relativa finestra di configurazione.



Per funzionare correttamente, SIA richiede che il controllo delle consistenze di GraphDB sia disattivato. Una caratteristica importante dei dataset situazionali è proprio la permissibilità di disaccordi in contesti isolati, ma GraphDB non tiene conto delle differenze di contesti per effettuare i controlli di consistenza.
Normalmente GraphDB controlla la presenza di inconsistenze durante l'inserimento e blocca l'intero inserimento nel caso una o più delle affermazioni inserite siano tra di loro incompatibili secondo qualche regola. Ci troviamo dunque costretti a fare a meno dei controlli predefiniti.
Al momento della creazione di una repository: è necessario verificare che i controlli di consistenza siano disattivati.

Sarà inoltre necessario selezionare il file di regole OWL2-RL-renamed, disponibile nella repository in allegato, per aggirare alcune scelte implementative relative ad owl:sameAs che interagiscono in maniera indesiderata con SIA. Ne specifichiamo meglio le ragioni nel capitolo successivo sui dettagli implementativi.



4 - Aspetti tecnici su SIA

SIA è un'estensione scritta in Kotlin per il triple-store GraphDB che ne sfrutta l'API per i plugin al fine di intervenire nella fase di Query Processing.
GraphDB informa SIA che alcuni pattern specifici sono stati inclusi dall'utente in una query SPARQL e SIA interpreta questi pattern ed effettua opportune operazioni.
L'operazione più ricorrente consiste nel creare contesti su cui effettuare inferenze locali.
I contesti creati in questo modo, e in generale le operazioni effettuate da SIA, non hanno effetti collaterali sulla repository. Tutte le operazioni sono effettuate su strutture dati mantenute in memoria e circoscritte alla query in questione.
Nel dettaglio, le inferenze locali vengono mantenute in memoria per la risoluzione della query ma non vengono materializzate nella repository. Le inferenze dovranno essere ripetute alla query successiva. La strategia applicata è quella di forward chaining descritta in precedenza.

SIA si appoggia al motore di inferenze di GraphDB ed è ignorante del modello di ragionamento utilizzato.[6] Si preoccupa esclusivamente di richiamare funzioni opportune del motore di inferenza, e sfrutta alcune callback nel ciclo di vita della richiesta per ottenere le proposizioni generate ed inserirle nel contesto corretto.

Il triple-store GraphDB™

GraphDB è un triple-store proprietario scritto in Java. Secondo [LTS], GraphDB è l'unico triplestore in grado di effettuare inferenze OWL su larga scala.
GraphDB permette di creare delle *repository* di triple RDF (più correttamente quadruple; hanno un contesto) e permette di selezionare il modello di inferenze desiderato.
Tra i modelli offerti compaiono frammenti di OWL sufficientemente espressivi per generare esempi interessanti. OWL 2 RL, già citato in precedenza, è stato selezionato perché in grado di individuare inconsistenze tra le triple inserite.
Il software è *closed-source* ma offre un SDK e un API per lo sviluppo di estensioni.
Diverse estensioni preesistenti sono rilasciate come software *open-source*.
La soluzione che andiamo a presentare deve necessariamente tenere conto di alcune limitazioni implementative dovute alle tecnologie scelte.

[6] In realtà, come vedremo, sono necessarie alcune accortezze di configurazione dovute a dettagli implementativi di GraphDB, ma l'algoritmo resta utilizzabile dato un qualsiasi ruleset.

Manteniamo però che, astraendo da questi dettagli, essa sia implementabile per altri triple-store e che sia indipendente dal modello di ragionamento.

Le regole di inferenza in GraphDB, e perché sono inadeguate

Le regole di inferenza sono definite secondo un formato particolare in un file .pie che viene compilato in artefatto Java per poi essere utilizzato da GraphDB.
Le regole consistono in una serie di premesse, a cui possono essere associate delle condizioni sul *pattern-matching*, e in una conclusione, che può essere composta da proposizioni o essere vuota.
Se le premesse di una regola sono soddisfatte e la conclusione contiene asserzioni, allora queste ultime verranno aggiunte alla *repository*. Se la regola ha una conclusione vuota, allora è una regola di consistenza; quando le sue premesse sono soddisfatte, c'è un'inconsistenza nella repository.
Le premesse sono nella forma "soggetto-predicato-oggetto-contesto". Ognuna di queste parti può essere una variabile.
Le conclusioni hanno una particolarità: i risultati che specificano un contesto di destinazione non sono visibili nella *repository* al momento di una query. Nel caso in cui il contesto compaia in una conclusione, siamo di fronte ad una regola intermedia, le cui triple generate saranno ulteriormente trattate da altre regole. [GDB-REASONING]
Questa scelta implementativa preclude la soluzione più immediata al problema delle inferenze situate. Non è possibile scrivere un file di regole che collochi le triple generate in alcuni contesti specifici.

Il plugin Provenance per GraphDB

SIA prende ispirazione dall'approccio del plugin Provenance per la realizzazione di inferenze locali nei contesti situati.
Il plugin Provenance[7] permette di raggruppare uno o più contesti in un contesto virtuale. Nel contesto virtuale, il plugin effettua inferenze locali trattando solo i contesti selezionati e le triple assiomatiche[8].
Il contesto viene definito all'interno della stessa query che vuole interrogarlo e le inferenze avvengono al momento della query.

[7] https://graphdb.ontotext.com/documentation/10.0/provenance-plugin.html
[8] Sono le triple definite dal regime di inferenza e tutte quelle inserite tramite Schema Transaction

Le triple all'interno del contesto così definito vengono rese disponibili alle interrogazioni SPARQL. Esse non sono materializzate all'interno del triple-store e vengono solamente mantenute in memoria per la durata della query.

Particolarità implementative dovute a limitazione tecniche dell'API

Un'obiezione più che giustificata che il lettore potrebbe sollevare è "perché effettuiamo le nostre inferenze al momento delle query e non durante l'insert".
Adottiamo questa strategia per via di come l'API di GraphDB ci permette di interagire con il ciclo di vita delle richieste SPARQL.
Va innanzitutto fatto notare che, per quanto le sintassi siano analoghe, il linguaggio di interrogazione "SPARQL 1.1 Query Language" è differente dal linguaggio di modifica "SPARQL 1.1 Update". I due sono definiti relativamente da [SPARQL-Q13] e [SPARQL-U13].
Non sappiamo come GraphDB implementi i due internamente, ma è ragionevole pensare che differenze di implementazione si siano riflettute sulla definizione dell'API.

Le query (e.g. SELECT, ASK, etc) sono gestite nella fase definita di "pattern interpretation" nel ciclo di vita del plugin. Tra le funzioni che ci permettono di interagire con questa fase, riportiamo la seguente, che viene richiamata almeno una volta per ogni statement SPARQL nella query.[9]

```
StatementIterator interpret(
        long subject, long predicate, long object, long context,
        pluginConnection PluginConnection,
        RequestContext requestContext
);
```

L'oggetto da notare è l'ultimo parametro: requestContext.
Il requestContext viene inizializzato nella prima sottofase del pattern interpretation e viene passato da GraphDB a tutte le chiamate di *interpret*. L'oggetto è utilizzato per mantenere lo stato durante le fasi di risoluzione della query e ci permette di accedere ai riferimenti di vari oggetti interni di GraphDB, tra cui l'*inferencer.*
Poter accedere all'inferencer è l'unico modo documentato di manipolare le inferenze di GraphDB ed è un punto centrale della nostra soluzione.

[9] Di fatto la funzione verrà richiamata più volte quando le variabili nella query diventano legate, e quando il meccanismo di ottimizzazione riordina gli statements nella query.

La fase di update interpretation (e.g INSERT, DELETE) , apparentemente analoga alla prima, non inizializza mai un requestContext. Di fatto l'oggetto inferencer non sembra essere accessibile e, al momento, non abbiamo individuato un modo di aggirare queste limitazioni.
Siamo costretti ad appoggiarci al ciclo di vita delle query e non degli update.

Un effetto positivo di lavorare in memoria al momento della query invece di materializzare le proposizioni all'inserimento è che non stiamo "inquinando" la repository.

Come SIA modifica il risultato delle query

Normalmente, un plugin per GraphDB ha modo di segnalare se è interessato o meno ad interpretare un certo pattern; forse aspetta predicati particolari e non deve interagire nella maggior parte delle query.
A parte rari casi, SIA tenterà sempre di gestire i pattern delle query.
Ad alcuni pattern specifici sono associati dei comportamenti particolari, ad esempio creare un contesto virtuale che rappresenta una situazione, ma spesso è opportuno che SIA intervenga nel modificare il risultato del matching di un generico pattern.
A titolo di esempio, se l'utente è interessato a tutti gli statements nella repository, sembra sensato aspettarsi che il risultato includa anche tutte le inferenze locali effettuate.
Di fatto, SIA prenderà in carico tutti i pattern di una query e tenterà di risolverli utilizzando i suoi contesti virtuali, se possibile.
Nel caso tipico, l'utente potrebbe aver dichiarato uno specifico contesto come situato e poi aver richiesto tutte le triple contenute in esso. SIA risponderà restituendo tutte le triple nel contesto virtuale. Si noti che di norma questo risulta nello *shadowing* del contesto vero e proprio: il contesto originario è ancora nella repository ma stiamo risolvendo la query utilizzando la nostra "copia" interna sulla quale abbiamo realizzato le inferenze.
Il contesto situato non può essere "obsoleto" rispetto a quello originario perché le inferenze sono sempre effettuate al momento della query e la fase di query è read-only.
Per evitare interazioni spiacevoli di SIA con altri plugins o con query speciali di GraphDb, è richiesto che la clausola `FROM` contenga `conj:situations`.

Come SIA modifica owl2-rl

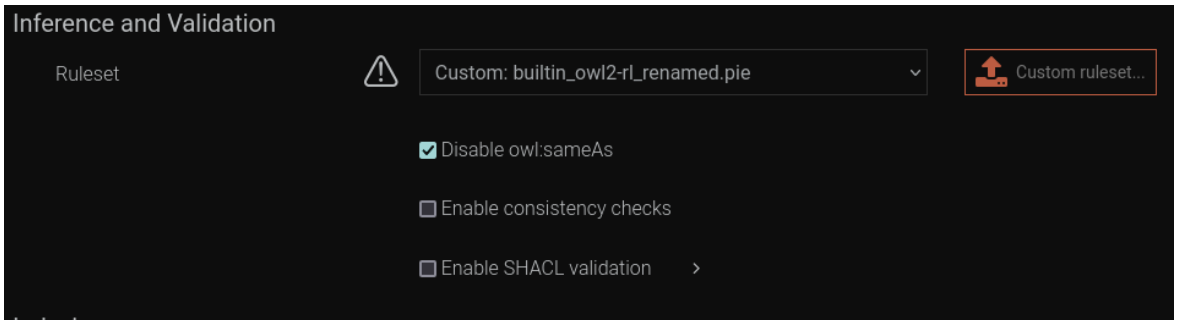
GraphDB ha un implementazione non standard di owl:sameAs.

Invece di usare regole ed assiomi come per tutti gli altri predicati, GraphDB fa hard-coding della semantica di owl:sameAs. Le regole sono riportate nel file .pie ma sono commentate.
Siccome owl:sameAs definisce una relazione di equivalenza (riflessiva, simmetrica e transitiva), GraphDB decide di rappresentare la classe di equivalenza come un singolo nodo [GDB-SAMEAS].
Di certo, questo è incompatibile con i nostri bisogni di circoscrivere l'applicazione delle regole di inferenza ad alcuni contesti.
Anche una volta decommentate, sembra che le regole relative ad owl:sameAs non vengano applicate normalmente nelle inferenze.
Sospettiamo che GraphDB riconosca le regole di owl:sameAs tramite id e che le ignori.
Riusciamo ad aggirare questa limitazione tecnica rinominando le regole.
Mostriamo di seguito un esempio:

```
Id: eq_sym
  s <owl:sameAs> o                                [Constraint s != o]
  -------------------------------
  o <owl:sameAs> s
```

Una volta rinominata in `eq_sym_conj` (un nome arbitrario), la regola viene trattata normalmente ed è utilizzabile nelle inferenze locali.

È anche necessario "disattivare" owl:sameAs alla creazione di una repository in modo che l'implementazione di GraphDb non disturbi le nostre inferenze.



Algoritmo di Calcolo della inferenze locali

Si parte da un insieme finito composto dalle triple assiomatiche e tutte le triple che l'utente ha dichiarato parte del contesto situato.
È dato un insieme finito di regole di inferenza che possiamo vedere come funzioni da p premesse a m triple inferite.
Per ogni tripla nell'insieme, per ogni regola p-aria, cerchiamo tutte le combinazioni di p elementi che includono la tripla. Se la combinazione soddisfa le premesse della regola, allora aggiungiamo le triple generate dalla regola all'insieme di partenza.
Il procedimento continua finché nuove triple sono generate.
Il tempo di convergenza varierà in base alla natura delle regole utilizzate.

Accortezze sulla ricerca dei disaccordi

Per trovare i disaccordi, verifichiamo quali tra le coppie di tutti i contesti consistenti sono inconsistenti se uniti.
L'algoritmo è dunque molto semplice, ma deve tenere conto di alcuni dettagli implementativi.
Ricordiamo che SIA funziona correttamente solo se il controllo di consistenza predefinito è disabilitato in quanto GraphDB non tiene conto dei contesti nei suoi controlli.
Per via di alcune scelte implementative di GraphDB, non siamo stati in grado di selezionare in maniera granulare quali proposizioni debbano essere considerate per il controllo e quali no.
Abbiamo aggirato questa limitazione creando delle repository virtuali, a tutti gli effetti delle piccole istanze di GraphDB, in cui il controllo di default delle inconsistenze è attivo. Nella repository vengono inserite una alla volta le triple dei due contesti. La repository in questione solleva un'eccezione in caso di inconsistenza e il plugin la cattura per comunicare all'utente che i due contesti sono in disaccordo.
Questa soluzione è sicuramente non ottimale. In primo luogo, la creazione di queste repository è costosa in termini di tempo e memoria (intorno alla quarta o quinta contemporanea, l'heap di java si riempie). In parte, riusciamo ad aggirare queste limitazioni *memoizando* i risultati dei controlli di consistenza.
Al momento, siamo costretti ad inserire le triple una per volta, il che comporta numerose transazioni e dunque overhead di tempo. Inoltre non siamo in grado di trovare tutte le asserzioni che causano un disaccordo. Infatti, l'eccezione viene sollevata alla prima inconsistenza e gli inserimenti non continuano.

Sospettiamo che sia possibile effettuare un controllo di consistenza utilizzando funzioni interne di GraphDB, ma al momento non siamo riusciti nell'intento.

File interessanti e classi principali

Quad.kt

La creazione di una classe Quad è stata tra le primissime astrazioni realizzate.
L'api di GraphDB rappresenta le quadruple in 3 modi diversi.

- Come `StatementIterator`, che contiene i campi `subject, predicate, object, context`
- Come `StatementIdIterator`, che contiene i campi sopra e un campo status addizionale che rappresenta alcuni flag interni, ad esempio se lo statement è implicito o esplicito
- Come Long Array, le cui posizioni rappresentano subject, predicate, object, context e status

La classe Quad ci permette di lavorare in modo uniforme senza preoccuparci della rappresentazione.

IteratorUtils.kt

Moltissime funzioni del plugin API di GraphDB si aspettano uno StatementIterator o uno StatementIdIterator.
Generalmente in questi casi preferiamo non lavorare direttamente su questi oggetti.
Preferiamo creare una `Sequence` di `Quad` che possiamo manipolare comodamente con funzioni lambda e, una volta processata, creare un iterator da restituire.

SituatedContext

È la classe che rappresenta un contesto situato. Ci permette di svolgere le inferenze localmente.
`mainContextId` è l'id del contesto da situare.
`AdditionalContexts` sono gli id di eventuali contesti di sharedKnowledge.
`SourceId` è utilizzato per generare le triple situanti. Generalmente coincide con `mainContextId.`

Supponiamo di voler situare il contesto con id 7. Potremmo creare un `SituatedContext` con id 47 e `sourceId` 7.

Nel restituire i risultati, per ogni tripla `subj pred 7`, dovremo restituire anche `subj pred 47`.

Come sono svolte le inferenze

Le funzioni importanti da notare sono `getStatementsToSituate`, `inferLocally`, `ruleFired` e `getRepStatements`

- getStatementsToSituate: prende le triple esplicite da `mainContextId` e dagli `AdditionalContexts`.
- inferLocally: applica la funzione AbstarctInferencer.doInference su tutte le triple.
  La funzione, definita da GraphDB, ha come callback ruleFired e getRepStatements, di cui possiamo fare override.
  Per ogni tripla e per ogni regola, inferLocally utilizzerà getRepStatements per tentare di soddisfare le premesse della regola corrente. Per ogni tripla generata da una regola soddisfatta, verrà chiamata ruleFired con la tripla passata come parametro.
- getRepStatements: definisce quali triple dobbiamo usare per tentare di soddisfare le premesse. Utilizziamo le triple assiomatiche nella repository e tutte le triple presenti nel contesto situato.
- ruleFired: in questa funzione decidiamo se inserire o meno una tripla generata nel contesto situato.

SituateTask

Ne viene creata un'istanza dal predicato conj:situateSchema.
Si preoccupa di creare e popolare i contesti situati definiti in uno schema.

SchemaForSituate

Rappresenta uno schema descritto dai triple pattern nella query che hanno contesto con prefisso `conj:schemas`
Viene accoppiato ad una task dal predicato conj:situateSchema.

Quotable

Quando creiamo un contesto situato a partire da un contesto normale, generalmente vogliamo che il contesto situato sia restituito in tutte triple che quotavano il contesto originario.
Questa Interfaccia descrive questo comportamento.
Un oggetto Quotable ha un QuotableId e un SourceId.
Il QuotableId identifica l'oggetto e interessa tutti gli altri oggetti che vogliono riferirsi ad esso.

Il SourceId è l'id del contesto di partenza. Per tutte le triple `SourceId pred obj` e `subj pred SourceId`, devono essere restituite le triple `QuotableId pred obj` e `subj pred QuotableId`, rispettivamente.

Reified

Un contesto è *reified* se contiene esclusivamente triple risultanti da reificazione standard.
Come tale, per tutte le triple S P O contenute, S è di tipo rdf:Statement.
Ci interessa isolare questo caso particolare perché dato un contesto Reified C, e data una tripla situante "subj pred C" o "C pred obj", vorremmo generare le triple "subj pred S" o "S pred obj", rispettivamente.
Le funzioni dichiarate devono implementare questo comportamento

Expandable

Questa interfaccia rappresenta un caso particolare di Quotable.
Dato un contesto C, per ogni tripla "a b c" in C, e per ogni tripla situate S P C o C P O, vorremmo generare le triple S P <<a b c>> o <<a b c>> P O, rispettivamente.
Le funzioni dichiarate devono implementare questo comportamento

SituatedInferenceContext

Da non confondere con `SituatedContext`.
`SituatedInferenceContext` è il contesto della query SPARQL. Mantiene lo stato delle operazioni del plugin.
Qui troviamo `inMemoryContexts`, che contiene tutti i contesti virtuali creati e gestiti dal plugin.
Troviamo anche le funzioni per la ricerca dei disaccordi.

5 - Valutazione

Valutazione qualitativa

Allo stato attuale, SIA non riesce pienamente a realizzare le funzionalità attese, ma resta, a nostro avviso, un utile primo tentativo che mostra una strategia valida per realizzare concretamente inferenze situate appoggiandosi a triple-store commerciali già esistenti.
Crediamo che la possibilità di dichiarare i contesti situati usando patterns RDF sia un meccanismo flessibile e di facile utilizzo.

Pensiamo che il tentativo di effettuare le conversioni al tempo di query sia stato un caso di errata *separation of concerns*. A posteriori, riteniamo che sarebbe stato più funzionale ed efficiente appoggiarci agli updates SPARQL.
Il proposito, a nostro avviso ben intenzionato, era di rendere i dettagli di conversione nascosti all'utente, che avrebbe semplicemente utilizzato una sintassi dichiarativa invece di preoccuparsi di scrivere manualmente gli updates. Avremmo anche preferito non materializzare le conversioni.
Crediamo che in una futura versione di SIA, compartimentalizzare le conversioni a delle funzioni SPARQL ad hoc possa essere un'alternativa valida per implementare le funzionalità desiderate con una sintassi più chiara e una buona usabilità.

Nella prossima sezione discutiamo le problematiche maggiori che restano presenti in SIA.
Sfortunatamente, ne abbiamo realizzato la gravità molto tardi nella fase di sviluppo, una volta guadagnata abbastanza esperienza e visione di insieme.

Analisi delle problematiche

Pattern interpretation e riordinamento delle query in GraphDb

Lanciata una query SPARQL, ogni plugin può dichiarare se intende interpretare un triple pattern nella query. Il primo plugin che dichiara interesse restituendo un risultato verrà utilizzato.
I plugin vengono informati sui valori e sulle variabili contenute dalle triple e, in base a queste informazioni, decidono come comportarsi.

Un plugin segnala interesse nell'interpretare una tripla ritornando un valore non-null. Successivamente, a seguito di stime sul numero di triple restituite da ogni pattern e in base alle variabili da legare, GraphDB riordina le triple. Se il valore legato di una variabile dovesse cambiare a causa del nuovo ordine, la tripla in questione verrà rivalutata.

Si intende facilmente che l'ordine delle triple come scritto nella query non corrisponderà all'ordine di valutazione finale.
Con un normale uso di SPARQL, questo non crea alcun problema: ogni pattern della query seleziona una porzione delle triple presenti nella repository e le triple disponibili sono le stesse durante tutta l'operazione. I risultati intermedi potrebbero avere dimensioni diverse, ma il risultato finale sarà lo stesso indipendentemente dall'ordine di valutazione.

Sfortunatamente, nella sua implementazione attuale, SIA è molto sensibile al riordinamento delle triple. I predicati di SIA si aspettano che alcune variabili siano legate, tentano di legarne altre e, nel frattempo, generano nuove triple non presenti nella repository.

Il seguente esempio mostra alcune delle problematiche:

Creare situazioni da triple RDF-STAR.

Supponiamo che il nostro dataset contenga le triple

```
:I :thinks << :Superman :can :clingFromCeiling >>.
:I :thinks << :Flash :can :clingFromCeiling >>.
```

Vorremmo creare il seguente contesto ed effettuare inferenze locali.

```
graph conj:I-thinks {
	:Superman :can :clingFromCeiling.
	:Flash :can :clingFromCeiling.
}
```

Dopo le inferenze:

```
graph conj:I-thinks-situated {
	:Superman :can :clingFromCeiling.
```

```
        :Flash :can :clingFromCeiling.
        :Superman rdf:type :SpiderSuperHero.
        :Flash rdf:type :SpiderSuperHero.
}
```

Abbiamo pensato la seguente query:

```
select distinct  ?subject ?predicate ?object from conj:situations where {
        :I :thinks ?triple.                             #1

        bind (conj:I-thinks as ?g)                      #2
        ?g conj:groupsTriple ?triple.                   #3

        graph conjSchemas:s {                           #4
                ?g a conj:SituatedContext.              #4a
                rdf4j:nil a conj:SharedKnowledgeContext.#4b
        }

        conj:task conj:situateSchema conjSchemas:s.     #5

        graph ?context {                                #6
                ?subject ?predicate ?object .           #7
        }
}
```

In #1, vorremmo prendere tutte le triple oggetto di `:I :thinks` e ci sembra sensato tentare di legarle a `?triple`.
Ora vorremmo che tutte queste triple finissero in un contesto virtuale, in modo da avere il nostro `conj:I-thinks` come descritto sopra.
Il predicato alla riga #3 prende le triple legate a `?triple` e crea un contesto virtuale `?g`.
Alla riga #2, stiamo fissando `conj:I-thinks` come nome di `?g`.
Notiamo subito che è essenziale che #1 sia eseguito prima di #3. Notiamo anche che #3 sta "creando" qualcosa in una query. Se l'ordine fosse invertito `conj:I-thinks` sarebbe vuoto.

In #4a vogliamo dichiarare `conj:I-thinks a conj:SituatedContext.` È necessario che #3 sia eseguito prima di #4a in quanto vogliamo che `?g` sia legata a `conj:I-thinks`.

Prima di arrivare alla tripla #5, che farà partire le inferenze locali, è importante che `conj:I-thinks` sia legato a `?g` (#2), che sia popolato (#3) e che sia dichiarato come situato (#4).

Il pattern #7 deve necessariamente essere l'ultimo, altrimenti le inferenze locali non saranno ancora state effettuate.
E in tutto questo non abbiamo ancora parlato della generazione delle triple situanti come
`:I :thinks << :Flash rdf:type :SpiderSuperHero >>!`

Crediamo che la presenza di tutte queste dipendenze e interazioni sia un sintomo chiaro di un'implementazione che deve essere almeno in parte ripensata.
In generale, SPARQL QUERY non è pensato per generare nuove triple. Tutti i pattern nella query di esempio sono da pensare come la descrizione della forma di un grafo che verrà risolta utilizzando le triple nella repository, mentre il nostro plugin usa i pattern analogamente a delle chiamate di funzione.
Stiamo usando un linguaggio dichiarativo pensato per effettuare pattern matching come se fosse un linguaggio procedurale.
L'idea è stata ispirata da esempi nella documentazione di GraphDB e da altre estensioni open source ma chiaramente non è adatta ai nostri scopi.

Fissare l'ordine delle triple

Per ogni pattern da interpretare, il plugin deve segnalare una stima del numero di triple che verrà restituito. Il numero di triple può dipendere dai valori e dalle variabili nella tripla. Riordinamenti diversi delle triple assegneranno diversi valori alle variabili e potrebbero comportare stime più o meno grandi. GraphDB riordina le triple utilizzando queste stime in modo da minimizzare la dimensione dei risultati intermedi.
SIA tenta di sfruttare (in maniera impropria) questo comportamento per fissare l'ordine delle triple.
Ad esempio #4a e #4b stimeranno di restituire una tripla; #5 ne stimerà 10. Di fatto l'utente potrebbe collocare le triple in ordine invertito ma il meccanismo di riordinamento provvederà a riposizionarle.

Questa soluzione, che sembrava promettente nelle prime fasi, si è rapidamente dimostrata contorta e difficile da mantenere. Con l'aumento dei predicati speciali e delle combinazioni è diventato impratico stabilire un ordine a priori.
L'idea si rompe del tutto quando andiamo a considerare dei pattern generici.
Nell'esempio precedente, vogliamo che `:I :thinks ?triple` sia risolto come primo pattern, ma ci troviamo in un caso molto specifico. In una query diversa, lo stesso pattern potrebbe essere l'ultimo, con l'intenzione di restituire tutti i valori compatibili con `?triple`.
Non è possibile distinguere i due casi a priori.

Retain Bind

Siamo a conoscenza della funzionalità Retain Bind di GraphDB che potrebbe in parte risolvere alcune delle problematiche riguardo il riordinamento delle triple.
Con "Retain Bind", anche se i pattern vengono riordinati, l'ordine delle variabili da legare non viene modificato. [GDB-BIND]
Riteniamo però che basarsi su un dettaglio implementativo molto caratteristico di GraphDB per rendere a malapena funzionante una soluzione mal pianificata non sia una scelta soddisfacente.

6 - Conclusioni e sviluppi futuri

Abbiamo iniziato la nostra discussione presentando le principali problematiche delle inferenze su dataset situazionali. Abbiamo anche menzionato le maggiori limitazioni degli approcci esistenti alle inferenze nell'ambito del Web Semantico.

Abbiamo introdotto il paradosso di Superman, che esemplifica una delle difficoltà tipiche del ragionamento su dataset situazionali. Alcuni formati di rappresentazione sono referenzialmente opachi per evitare istanze di problematiche simili. Abbiamo notato che il paradosso è riconducibile ad un problema più esteso riguardo l'uso inopportuno di regole e antecedenti nelle inferenze situate.

Vengono proposte le 3 assunzioni di razionalità, permeazione, e località, con lo scopo di effettuare inferenze più coerenti su dataset situazionali. In questi dataset, le asserzioni situanti possono essere viste come contesti che determinano il valore di verità delle asserzioni situate.

Partendo dalle 3 assunzioni discusse, viene introdotto il Situated-Inferences-Addon (SIA), con l'intento di effettuare inferenze situate senza paradossi su triple-store commerciali.
La versione di SIA discussa in questa dissertazione non implementa ancora tutte le funzionalità desiderate, ma riesce a mostrare alcuni risultati utili.
Restringendo le inferenze alle proposizioni interne ai contesti, e limitando la validità dei risultati ai contesti stessi, riusciamo ad evitare istanze del paradosso di Superman.
La soluzione non si appoggia a specifiche regole di inferenza, dunque riteniamo che sia implementabile in triple-store e utilizzabile con altri modelli di ragionamento.

Per quanto ci risulta, al momento non esistono alternative a SIA che utilizzino lo stesso approccio per la realizzazione di inferenze situate. In [NS17], gli autori implementano una soluzione basata sulle Singleton Properties.

Nella sua iterazione attuale, SIA gioverebbe di numerosi miglioramenti e non ci sentiamo di consigliarne l'utilizzo fino a che alcune problematiche non saranno risolte.
In primo luogo, restano vari comportamenti inattesi dovuti all'uso improprio dei pattern SPARQL e alla loro interazione con il meccanismo di binding e riordinamento.

Riconosciamo anche l'impraticità della sintassi proposta, che non è coerente con il normale uso di SPARQL.
Al momento il meccanismo di conversione di RDF-Star resta non del tutto funzionante.
Ci aspettiamo anche limitazioni in efficienza e scalabilità; di certo non è possibile effettuare ripetutamente inferenze al tempo di query con velocità ragionevoli quando il numero di triple cresce.

Di seguito discutiamo in ordine di priorità le modifiche che intendiamo apportare a SIA.
Per prima cosa, riteniamo sensato modificare la strategia di dichiarazione dei contesti situati che al momento avviene tramite alcuni pattern speciali nella query. Sospettiamo che il riordinamento delle query possa risultare diverso tra l'ambiente di test e GraphDB; in generale, non possiamo aspettarci che sia stabile tra una versione e l'altra.[10]
Nella sua iterazione attuale, SIA è molto sensibile all'ordine delle triple nelle query.
Cercare di interagire con i meccanismi di binding e alcune peculiarità della sintassi RDF-Star ha dato vita ad un meccanismo contorto e dalla dubbia solidità.
Per motivazioni simili, il meccanismo di conversione, che al momento è gestito da predicati speciali, soffre dei medesimi problemi.
Crediamo più corretto separare chiaramente le operazioni di conversione delle triple e di dichiarazione delle situazioni dalle operazioni di interrogazione.
Una soluzione di questo tipo sfrutterebbe in maniera coerente le funzionalità di SPARQL QUERY e SPARQL UPDATE.
In futuro, l'utente dichiarerà i contesti inserendo specifiche triple attraverso update.
Le conversioni verranno implementate attraverso funzioni SPARQL[11] che l'utente potrà usare sia nel pattern matching, che nelle proiezioni (clausola SELECT) e negli aggiornamenti (UPDATE).
La clausola WHERE degli update offre molto controllo all'utente e riteniamo sensato appoggiarci ad essa ed alleggerire il lavoro si pattern interpretation di SIA.
In fase di query, SIA consulterà la repository per stabilire quali contesti sono dichiarati come situati.
Sarà probabilmente necessario mantenere un predicato con una semantica particolare che ordini al plugin di effettuare le inferenze locali durante una query, ma un solo predicato senza interazioni è facilmente gestibile e, di fatti, già implementato.

[10] A parità di configurazione, abbiamo notato differenze nei risultati tra la versione 10.1 e 10.3
[11] https://rdf4j.org/documentation/tutorials/custom-sparql-functions/

Facciamo notare che, al momento, non c'è un modo conciso per creare *tutte* le situazioni determinate da triple situanti espresse in RDF-star o reificate. Con le funzioni SPARQL, il comportamento desiderato si ottiene in poche righe, L'utente utilizzerà le funzioni per effettuare le conversioni in fase di update e SIA interverrà per effettuare le inferenze e cercare i disaccordi.

Mostriamo un esempio di un'operazione di UPDATE SPARQL che raggruppa tutte le quoted triples rdf-star situate dalla stessa tripla.
L'operazione si può astrarre in una funzione SPARQL.
Iniziamo con il seguente update:

```
PREFIX : <http://a#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT DATA {
	:LoisLane :thinks <<:Superman :can :fly>>.
	:LoisLane :thinks <<:Superman rdf:type :Superhero>>.
	:I :thinks <<:Pizza :is :tasty>>
}
```

Eseguiamo un altro update con pattern matching:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX conj: <https://w3id.org/conjectures/>
PREFIX : <http://a#>
PREFIX rdf4j: <http://rdf4j.org/schema/rdf4j#>
PREFIX spif: <http://spinrdf.org/spif#>
PREFIX afn: <http://jena.apache.org/ARQ/function#>

INSERT {
	graph :thoughts {
		rdf4j:nil a conj:SharedKnowledgeContext.
		?g a conj:SituatedContext .
	}
	graph ?g {
		?ss ?sp ?so
```

```
        }
} WHERE {
        BIND (<<?ss ?sp ?so>> AS ?triple)
        {       ?a ?b ?triple        }
        UNION
        {       ?triple ?a ?b        }
        BIND  (spif:buildURI("<conj:{?1}-{?2}>",  afn:localname(?a),  afn:localname(?b)  )
as ?g)
}
```

Otteniamo i seguenti risultati senza scomodare il plugin.

	s	p	o	g
1	:Superman	rdf:type	:Superhero	conj:LoisLane-thinks
2	:Superman	:can	:fly	conj:LoisLane-thinks
3	conj:LoisLane-thinks	rdf:type	https://w3id.org/conjectures/SituatedContext	:thoughts

	s	p	o	g
1	:Pizza	:is	:tasty	conj:I-thinks
2	conj:I-thinks	rdf:type	https://w3id.org/conjectures/SituatedContext	:thoughts

A questo punto SIA può facilmente effettuare inferenze locali su questi contesti visto che sono già nella repository.

In maniera analoga, è possibile trovare esempi di SPARQL usato per la conversione da reificazione standard a rdf-star e viceversa come mostrato in [GDB-CONV]

Per quanto riguarda il controllo delle consistenze, sospettiamo che sia possibile effettuare inserimenti in blocco nelle repository temporanee di lavoro. Questo renderebbe più efficienti i controlli e ci permetterebbe di ottenere tutte le triple in disaccordo.
Siamo comunque alla ricerca di una soluzione che ci permetta di evitare l'overhead e la complicazione di gestire le repository temporanee.

Una funzionalità di bassa priorità ma che desideriamo implementare è la ricerca degli antecedenti per inferenze situate. Il plugin Proof per GraphDB[12] effettua la ricerca degli antecedenti di una tripla ma non tiene conto del contesto di provenienza della triple. Vorremmo adattare il plugin al nostro caso per permettere all'utente di stabilire quali triple e quale regola in un contesto situato hanno generato una certa proposizione situata.

[12] https://graphdb.ontotext.com/documentation/10.0/proof-plugin.html

7 - Bibliografia

[ABF19] Sahar Aljalbout, Didier Buchs, and Gilles Falquet. Introducing contextual reasoning to the semantic web with OWL. In Dominik Endres, Mehwish Alam, and Diana Sotropa, editors, Graph-Based Representation and Reasoning - 24th International Conference on Conceptual Structures, ICCS 2019, Marburg, Germany, July 1-4, 2019, Proceedings, volume 11530 of Lecture Notes in Computer Science, pages 13–26. Springer, 2019.

[BGvH+03] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-owl: Contextualizing ontologies. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, The Semantic Web - ISWC 2003, pages 164–179, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[FFP+09] Giorgos Flouris, Irini Fundulaki, Panagiotis Pediaditis, Yannis Theoharis, and Vassilis Christophides. Coloring RDF triples to capture provenance. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings, volume 5823 of Lecture Notes in Computer Science, pages 196–212. Springer, 2009

[For96] Forrest, Peter. “The Identity of Indiscernibles (Stanford Encyclopedia of Philosophy).” Stanford Encyclopedia of Philosophy, 31 July 1996, https://plato.stanford.edu/entries/identity-indiscernible/. Accessed 18 September 2023.

[GDB-BIND] “Retain BIND Position Special Graph — GraphDB 10.1.0 documentation.” *GraphDB*, 6 March 2023, https://graphdb.ontotext.com/documentation/10.1/retain-bind-position-special-graph.html. Accessed 30 September 2023.

[GDB-CONV] “RDF-star and SPARQL-star — GraphDB 10.0.0 documentation.” *GraphDB*, 6 March 2023, https://graphdb.ontotext.com/documentation/10.0/devhub/rdf-sparql-star.html#convert-standard-reification-to-rdf-star. Accessed 30 September 2023.

[GDB-REASONING] “Reasoning — GraphDB 10.1.0 documentation.” GraphDB, 6 March 2023, https://graphdb.ontotext.com/documentation/10.1/reasoning.html#. Accessed 29 September 2023.

[GDB-SAMEAS] “Optimization of owl:sameAs — GraphDB 10.0.0 documentation.” GraphDB, 6 March 2023, https://graphdb.ontotext.com/documentation/10.0/sameas-optimisation.html#optimization-of-owl-sameas. Accessed 27 September 2023.

[LTS] “LargeTripleStores.” W3C, https://www.w3.org/wiki/LargeTripleStores#GraphDB.E2.84.A2_by_Ontotext_.2817B.29. Accessed 29 September 2023.

[Lup23] Luper, Steven. “Epistemic Closure (Stanford Encyclopedia of Philosophy).” Stanford Encyclopedia of Philosophy, 31 December 2001, https://plato.stanford.edu/entries/closure-epistemic/. Accessed 15 September 2023.

[N3] “Notation3 (N3): A readable RDF syntax.” W3C, 28 March 2011, https://www.w3.org/TeamSubmission/n3/. Accessed 29 September 2023.

[Nel23] Nelson, Michael. "Propositional Attitude Reports." The Stanford Encyclopedia of Philosophy, Spring 2023 Edition, edited by Edward N. Zalta & Uri Nodelman, URL: https://plato.stanford.edu/archives/spr2023/entries/prop-attitude-reports/. Accessed 29 September 2023.

[Forbes] Forbes, Graeme. “Referential Opacity.” *Internet Encyclopedia of Philosophy*, https://iep.utm.edu/referential-opacity/. Accessed 30 September 2023.

[NS17] Vinh Nguyen and Amit P. Sheth. Logical inferences with contexts of RDF triples. CoRR, abs/1701.05724, 2017

[NV14] Nguyen, Vinh et al. “Don't Like RDF Reification? Making Statements about Statements Using Singleton Property.” Proceedings of the ... International World-Wide Web Conference. International WWW Conference vol. 2014 (2014): 759-770. doi:10.1145/2566486.2567973

[OWL] “OWL - Semantic Web Standards.” W3C, https://www.w3.org/OWL/. Accessed 29 September 2023.

[Pat14] Patel, Peter F. “RDF 1.1 Semantics.” W3C, 25 February 2014, https://www.w3.org/TR/rdf11-mt/. Accessed 13 September 2023.

[Pru12] Eric Prud’hommeaux. Untitled, 04 2012. https://www.w3.org/2001/12/attributions/.

[RDF-C14] “RDF 1.1 Concepts and Abstract Syntax.” W3C, 25 February 2014, https://www.w3.org/TR/rdf11-concepts/#dfn-named-graph. Accessed 28 September 2023.

[RDF-DS14] “RDF 1.1: On Semantics of RDF Datasets.” W3C, 25 February 2014, https://www.w3.org/TR/rdf11-datasets/#the-graph-name-denotes-the-named-graph-or-the-graph. Accessed 13 September 2023.

[RDF-MT04] “RDF Semantics.” *W3C*, 10 February 2004, https://www.w3.org/TR/rdf-mt/. Accessed 14 September 2023.

[RDF-STAR23] “RDF-star and SPARQL-star.” W3C on GitHub, 29 June 2023, https://w3c.github.io/rdf-star/cg-spec/editors_draft.html#background-and-motivation. Accessed 11 September 2023.

[SPARQL-Q13] “SPARQL 1.1 Query Language.” W3C, 21 March 2013, https://www.w3.org/TR/sparql11-query/. Accessed 22 September 2023.

[SPARQL-U13] “SPARQL 1.1 Update.” W3C, 21 March 2013, https://www.w3.org/TR/sparql11-update/. Accessed 22 September 2023.

[SW-FAQ] Herman, Ivan. “W3C Semantic Web FAQ.” W3C, https://www.w3.org/RDF/FAQ. Accessed 11 September 2023.

[TRIG14] “RDF 1.1 TriG.” *W3C*, 25 February 2014, https://www.w3.org/TR/trig/. Accessed 16 September 2023.

8 - Allegati

Link alla repository github di Situated-Inferences-Addon:
https://github.com/andrea-corradetti/situated-inferences/tree/Situation