OwLogics

Inferenze Logiche con Prolog

PRESENTAZIONE DI
GIORGIO BERNASCONI E
ALESSIO FARIOLI

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

DESCRIZIONE

Esploriamo il progetto

- **Analisi con Prolog**: Analisi di dataset Turtle per sfruttare le capacità di inferenza di Prolog.
- **Parser dedicato**: Traduzione di dataset Turtle in formato Prolog, assicurando corretta **formattazione e gestione errori**.
- **Inferenza di Relazioni**: Utilizzo di RDFS e OWL per inferire relazioni, classi e proprietà da triple.
- **Gestione Relazioni e Gerarchie**: Gestione accurata di classi, individui, proprietà per **coerenza e ampliamento**.
- **Arricchimento Dataset**: Ampliamento delle possibilità interrogative tramite inferenze avanzate.

Obiettivo

Implementare in Prolog un sistema per fare **inferenze e Query** su dataset Turtle, ampliando le possibilità di interrogazione

TURTLE

Trutle per il dataset

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<Person> rdf:type owl:Class .
<Animal> rdf:type owl:Class .
<Mammal> rdf:type owl:Class .
<Dog> rdf:type owl:Class .

<Mammal> rdfs:subClassOf <Animal> .
<Dog> rdfs:subClassOf <Mammal> .

<hasPet> rdf:type owl:ObjectProperty .
<hasOwner> rdf:type owl:ObjectProperty .

<hasPet> rdfs:domain <Person> .
<hasPet> rdfs:range <Animal> .
<hasOwner> rdfs:domain <Animal> .
<hasOwner> rdfs:range <Person> .

<John> rdf:type <Person> .
<Fido> rdf:type <Dog> .

<John> <hasPet> <Fido> .
<Fido> <hasOwner> <John> .
```

Cos'è Turtle?

Turtle (Terse RDF Triple Language) è un **formato di serializzazione** per i dati RDF

- **Leggibilità e Concisità:** Turtle offre una sintassi intuitiva e concisa, migliorando la **leggibilità** per umani e **facilità di elaborazione** da parte dei computer.
- **Uso di Prefissi:** Introduce l'utilizzo di **prefissi** per abbreviare gli URI, rendendo la notazione più **semplice** e i dati più **accessibili**.
- **Strutture di Dati Complesse:** Supporta l'espressione diretta e semplificata di **strutture** di dati **complesse**, come collezioni e liste ordinate.

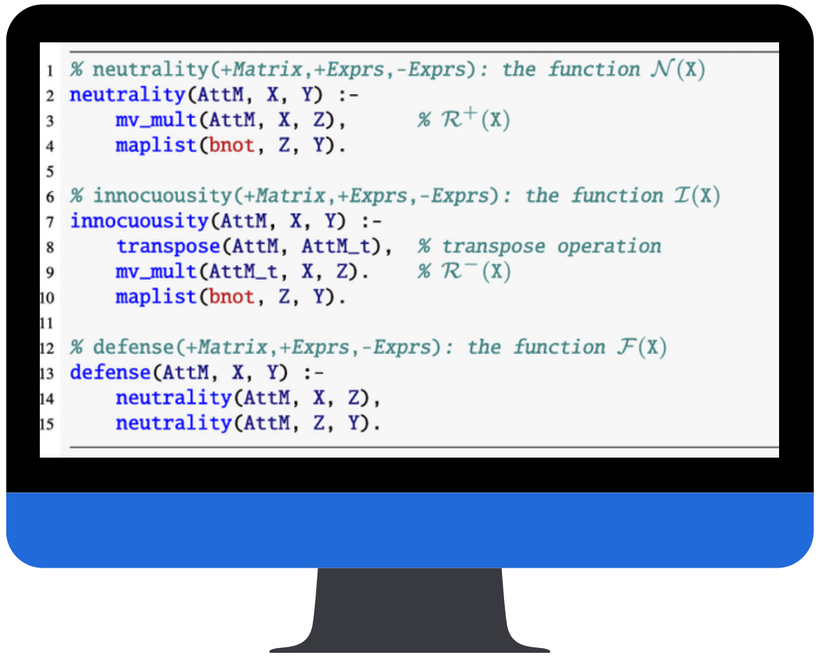
Perchè Turtle?

- **Semplicità e Efficienza:** Turtle **semplifica** la **rappresentazione** RDF, rendendola più efficiente e meno verbosa rispetto ad altri formati.
- **Leggibilità Migliorata:** Grazie all'uso di **prefissi**, Turtle facilita la lettura e la scrittura, evitando la ripetitività degli IRI lunghi.
- **Ampio Supporto:** Supportato ampiamente nell'ecosistema RDF, Turtle è diventato uno **standard pratico** per lo sviluppo Semantic Web.

PROLOG

Perchè prolog?

Prolog è un linguaggio di programmazione logica incentrato sulla rappresentazione di fatti e regole, facilitando interrogazioni e inferenze complesse.



```
% neutrality(+Matrix,+Exprs,-Exprs): the function 𝒩(X)
neutrality(AttM, X, Y) :-
    mv_mult(AttM, X, Z),      % ℛ⁺(X)
    maplist(bnot, Z, Y).

% innocuousity(+Matrix,+Exprs,-Exprs): the function ℐ(X)
innocuousity(AttM, X, Y) :-
    transpose(AttM, AttM_t),  % transpose operation
    mv_mult(AttM_t, X, Z).    % ℛ⁻(X)
    maplist(bnot, Z, Y).

% defense(+Matrix,+Exprs,-Exprs): the function ℱ(X)
defense(AttM, X, Y) :-
    neutrality(AttM, X, Z),
    neutrality(AttM, Z, Y).
```

SWI Prolog

Motivi della sceltà

- **Rappresentazione Naturale**: Prolog consente una **mappatura** diretta delle **triple RDF,** rendendo la rappresentazione dei dati RDF intuitiva e gestibile.
- **Potente Interrogazione e Inferenza:** Grazie al suo sistema di regole, Prolog permette di **eseguire query complesse** e di dedurre nuove informazioni, **allineandosi** perfettamente con gli **obiettivi di RDF**.
- **Enfasi sulla Semantica:** Entrambi, Prolog e RDF, si **concentrano** sulla **logica e sulla semantica**. Prolog utilizza questa capacità per manipolare semanticamente i dati RDF, interpretando le informazioni in modo significativo.
- **Flessibilità ed Estendibilità:** Prolog offre la libertà di definire regole su misura e può essere adattato per **includere funzionalità specializzate**, rendendolo estremamente **versatile** per lavorare con RDF.

PANORAMICA GENERALE

Come si è svolto il progetto

Esaminiamo l'evoluzione del progetto: da Turtle e RDFS all'integrazione di OWL e miglioramenti nel parsing, verso un sistema di inferenza più avanzato

Da turtle a triple/3

- **Struttura Turtle**: Analisi di Turtle basata su righe di **quattro elementi** - Soggetto, Verbo, Oggetto, Separatore.
- **Gestione Errori**: Implementazione della gestione degli **errori di formato** per garantire la correttezza dei dati.

QUERY

- **Sinergia con il Parser e Funzionalità di Query**: Abbiamo **integrato le due parti**. Abbiamo inoltre implementato funzioni per le query, inclusa la possibilità di utilizzare il comando **Select**/3 per interrogazioni mirate e per una maggiore praticità

Gestione del dataset e inferenze

- **Implementazione Base RDFS**: Inizialmente, il programma è stato configurato per supportare le **funzionalità fondamentali** di RDFS (classi, sottoclassi, e proprietà) per una **solida base**
- **Evoluzione verso OWL:** Abbiamo poi esteso le capacità del programma includendo le **caratteristiche di OWL**. Nuove funzioni per supportare la logica più complessa di **equivalenza** di classi, **disgiunzione**, e **restrizioni** su proprietà, arricchendo l'analisi e inferenza.

ANALISI PROGETTO

Entriamo nel dettaglio

Parser

il progetto si occupa di implementare in prolog un **purser di turtle**. Il parser si occupa di **gestire i prefissi**, tramite **prefix/2**, sostituendoli nel contesto in cui vengono usati. Vengono gestiti i casi di semplice asserzione e casi più complesssi.
Realizza in modo **ricorsivo** riga per riga. Si può scegliere un nome del file, con nome **assoluto e relativo**, a seconda se usi read_abs_file oppure read_rel_file.

Collaborazione tra i due

- Si utilizza il parser per caricare il file di input e **tradurlo**.
- Successivamente con **start_execution** si avvia il **reasoning** per generare nuove triple.
- Una volta concluso la funzione **stampa** in automatico tutto il **dataset aggiornato**.
- Grazie a **Select/3** è possibile fare **Query**.

Sistema di inferenze

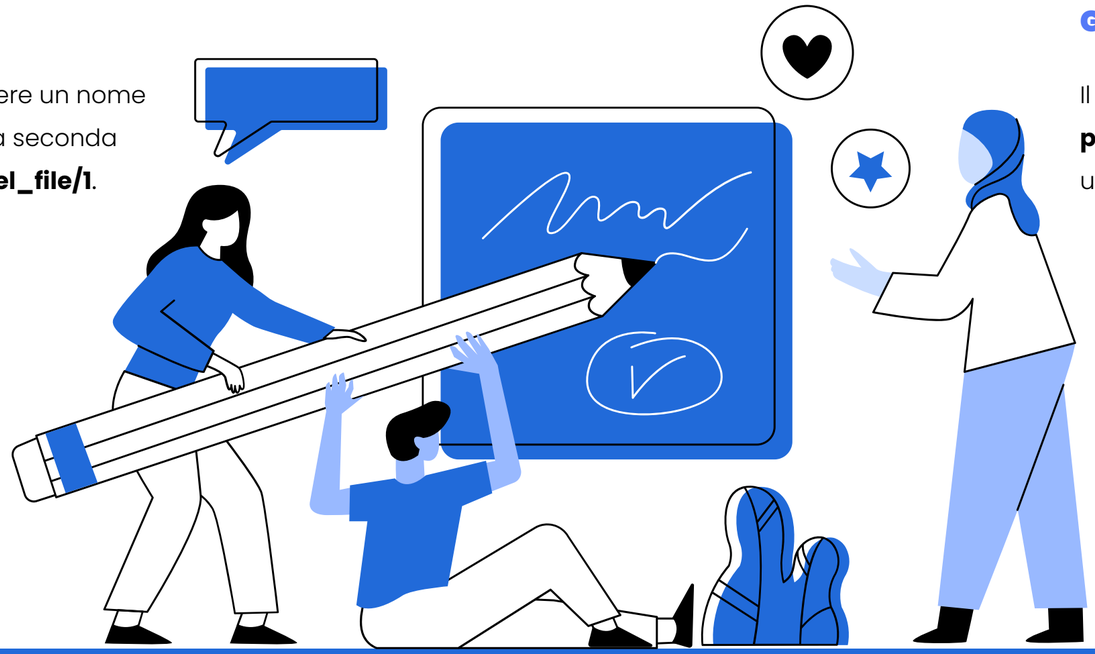
Il nostro sistema di inferenza utilizza **regole logiche** basate su **OWL** e **RDFS** per arricchire il dataset con nuove relazioni e proprietà. **Analizzando** le strutture esistenti, sulla base delle loro **propietà** e **caratteristiche**, inferisce nuove triple per migliorare la capacità di query. Esplora proprietà come bidirezionalità e transitività per ampliare la conoscenza implicita. E' presente un meccanismo per **rilevare inconsistenze** nel dataset.

Lettura

Prende in input un file .**ttl**. Si può scegliere un nome del file, con nome **assoluto e relativo**, a seconda se usi **read_abs_file/1** oppure **read_rel_file/1**.

Esecuzione

Scansiona ricorsivamente il file di input e effettua la **traduzione** in formato prolog, utilizzando **triple/3**.



Gestione prefissi

Il parser si occupa di **gestire i prefissi**, tramite **prefix/2**, sostituendoli nel contesto in cui vengono usati.

Casi gestiti

Semplice asserzione, tripla con **più oggetti** (separate da una virgola, purché sulla stessa riga), casi con **più verbi** per stesso soggetto (separati da ; su righe diverse e indentati tramite "tab"), sono accettati **commenti** (purché su una riga e con #) e **righe vuote** (completamente vuota quindi senza spazi).

FOCUS ON WORK

R D F S

RDFS nel nostro progetto

Funge da strumento chiave per la modellazione delle ontologie e la descrizione semantica dei dati nel Web Semantico, migliorando la leggibilità e l'elaborabilità delle informazioni.

01

Descrizione

RDFS (RDF Schema) **arricchisce RDF** stabilendo una base per la gerarchia delle classi e la tipizzazione delle proprietà.

02

USO

Il nostro sistema **identifica** le triple contenenti elementi come "**rdfs:Class**" e "**rdfs:Property**", basandosi su deduzioni logiche, **inferisce nuove triple**, arricchendo così la base di conoscenza disponibile.

03

Esempi

```
is_subClass/2, infer_class_membership/0,
infer_transitive_subbClass/0,
apply_domain_and_range_restrictions /3,
nothing_hierarchy/0
```

LIMITAZIONI

Limitazioni di RDFS

RDFS non basta

- **Limiti di Inferenza**: RDFS si limita a **inferenze semplici** senza supportare regole logiche avanzate o restrizioni di classe, **restringendo** la profondità delle **interrogazioni semantiche**.
- **Mancanza di Espressività**: RDFS mostra **limiti** nell'**esprimere proprietà** avanzate come **transitività** e restrizioni di **simmetria**, cruciali per descrivere relazioni complesse tra entità.
- **Impatto sul Progetto**: Queste restrizioni **limitano** le nostre **capacità** di modellazione e interrogazione, influenzando l'efficacia nella gestione di dataset semantici e la generazione di nuove intuizioni.

OWL

Web Ontology Language, offre una soluzione alle limitazioni di RDFS, introducendo un livello di **espressività** e **capacità inferenziale** significativamente **maggiore**. A differenza di RDFS, OWL permette la definizione di classi, proprietà e relazioni con un **grado di dettaglio** molto più **ricco**, supportando concetti come: Classi Equivalenti e Disgiunte, Proprietà Avanzate, ecc..

OWL

Superare le limitazioni con OWL

Cos'è OWL:

OWL (Web Ontology Language) è una potente **estensione di RDFS** progettata per **incrementare l'espressività** nella modellazione ontologica. Fornisce **strumenti avanzati** per definire in modo dettagliato classi, proprietà e le loro interrelazioni, consentendo una **rappresentazione** dei dati semantici più **ricca** e complessa.

Espansione delle Capacità di Inferenza:

Il nostro sistema guadagna la capacità di effettuare **inferenze logiche** più **sofisticate**. Questo include la deduzione di relazioni, come identificare automaticamente le **classi equivalenti**, gestire **proprietà transitive** e il controllo di coerenza (**consistency check**),grazie al quale possiamo rilevare ed **eliminare le inconsistenze** nel dataset.

Esempi

```
infer_symmetric_property/0,
infer_equivalent_classes/0,
infer_disjoint_individuals/0
```

OWL

Considerazioni su owl

L'adozione di OWL segna un progresso significativo nelle capacità inferenziali del nostro sistema, ampliando la modellazione e l'analisi dei dati oltre i confini di RDFS.

C'è un "MA"

Tuttavia, riconosciamo che la nostra implementazione attuale sfrutta **solo** una **frazione** delle **potenzialità** di OWL. Mentre abbiamo fatto passi da gigante nell'utilizzo di alcune delle sue funzionalità avanzate, esiste un **ampio spazio** per **incorporare** ulteriori **aspetti** di OWL, promettendo un'evoluzione continua verso un sistema di inferenza ancora **più completo e sofisticato**.

Queste rappresentano **aggiunte teoriche** al nostro codice, esplorando argomenti ulteriori rispetto a quelli trattati a lezione.

PIANI FUTURI

Ulteriori funzionalità di OWL

- **Classi Composite**: Attraverso **owl:intersectionOf** e **owl:unionOf**, possiamo creare classi che rispondono a condizioni logiche complesse, per una **modellazione** più **dettagliata**.
- **Restrizioni di Proprietà**: Utilizziamo **owl:cardinality** e **owl:allValuesFrom** per imporre vincoli precisi sulle relazioni, **migliorando la specificità** dei dati.
- **Proprietà Avanzate**: L'adozione di proprietà come **owl:AsymmetricProperty** e **owl:InverseFunctionalProperty** permette di rappresentare relazioni specifiche tra entità.
- **Classi Disgiunte**: Con **owl:AllDisjointClasses**, definiamo **insiemi di classi** esclusive per una **categorizzazione chiara** e non sovrapposta.

PROBLEMATICHE

Alcuni imprevisti e situazioni particolari incontrate

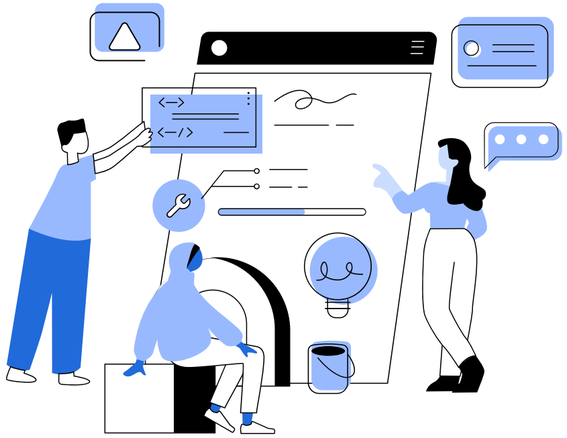
Incconsistenze

Nel nostro progetto la **gestione** delle **inconsistenze** è una parte molto importante, ma nella realizzazione del progetto abbiamo dovuto valutare come gestire le **contraddizioni**. Abbiamo ritenuto fosse **ottimale eliminare** ambedue **le triple** che causano contraddizione in quanto non ne esiste una più corretta dell'altra

Prefissi

Nell'integrazione tra parser e reasoner abbiamo incontrato alcuni problemi dato che sostituiamo gli IRI al posto dei prefissi. Ciò era **ottimo** per il **controllo** della **formattazione** ma per l'**output del parser** e per le **Query non** era per nulla **desiderabile**. Abbiamo optato per creare una **funzione inversa** durante gli output.
Un altra soluzione sarebbe potuta essere la modifica alla fine del parsing ma ciò avrebbe richiesto notevoli modifiche al codice

DISCUSSIONE FINALE



Prolog

Siamo riusciti nella **realizzazione** con successo del parser. E' in grado di leggere file "**.ttl**" sia come assoluti che relativi.

Reasoner

Il reasoner **implementa** le propietà di **RDFS** perfettamente.
Abbiamo inoltre **applicato** diverse **propietà** di **OWL**.

Punti di miglioramento

OWL **introduce** moltissime possibilità in un progetto di questo tipo, pertanto esistono **ulteriori casistiche** che ancora potrebbero essere gestite.