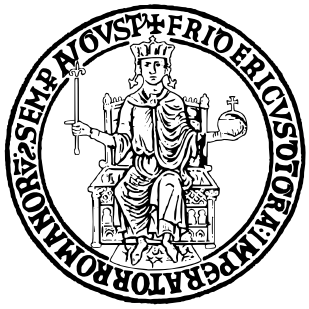
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Magistrale in Informatica

Progetto Intelligent Web

Professore
Luigi Sauro

Matricola
Jonathan Quaranta N97/0413
Antonio Nardella N97/0412
Alessandro Schiavo N97/423

Anno Accademico 2023–2024

Contents

1 Introduzione

Il documento corrente dettaglia il processo di sviluppo di un reasoner specificamente progettato per una variante semplificata della logica descrittiva EL++, seguendo le indicazioni fornite nell'articolo "Pushing the EL Envelope" di Franz Baader, Sebastian Brandt e Carsten Lutz. Questo lavoro ha il fine di analizzare la sussunzione tra due concetti all'interno di una base di conoscenza, formulata come (KB $\models A \sqsubseteq B$), utilizzando costrutti descritti dettagliatamente nell'articolo in termini di sintassi e semantica.

Il progetto si è concentrato sulla progettazione e implementazione di un reasoner sviluppato in Java attraverso l'utilizzo delle OWLAPI, con il reasoner che implementa un metodo per la verifica di sussunzione tra due concetti: (KB $\models C \sqsubseteq D$) dopo aver trasformato le entità KB, C e D in forma normale e classificato i concetti atomici secondo le regole di completamento CR1-CR6, come richiesto dalla traccia.

Inoltre, la parte fondamentale del progetto è stata dedicata al testing funzionale, svolto attraverso l'ambiente Protegè, per validare l'efficacia del reasoner e per testarne le prestazioni con vari scenari di query.

2 Progettazione

2.1 Descrizione di EL++ e Analisi del problema

La logica descrittiva EL++ rappresenta un'estensione della logica EL e permette la manipolazione di concetti utilizzando principalmente l'operatore esistenziale e la congiunzione. In realtà la logica EL++, oltre ad includere gli operatori di congiunzione e quello esistenziale introduce ulteriori costrutti che ampliano le sue capacità. Tuttavia, alcuni di questi costrutti non sono supportati dal reasoner che è stato sviluppato. Di conseguenza, le concept description che richiedono tali costrutti non sono gestibili dal nostro reasoner. Nella seguente tabella vengono illustrate struttura e sintassi della logica EL++:

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
concrete domain	$p(f_1,\ldots,f_k)$ for $p \in \mathcal{P}^{\mathcal{D}_j}$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y_1,\ldots,y_k \in \Delta^{\mathcal{D}_j} : f_i^{\mathcal{I}}(x) = y_i \text{ for } 1 \leq i \leq k \wedge (y_1,\ldots,y_k) \in p^{\mathcal{D}_j}\}$
GCI	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
RI	$r_1 \circ \cdots \circ r_k \sqsubseteq r$	$r_1^{\mathcal{I}} \circ \cdots \circ r_k^{\mathcal{I}} \subseteq r^{\mathcal{I}}$

Figura 1.1: EL++ (sintassi e semantica)

Dal documento "Pushing the EL Envelope", si evince che data una base di conoscenza C, l'insieme BC_c è il più piccolo insieme che contiene il Top, tutti i nomi di concetti utilizzati nella base di conoscenza e tutti i nomi di individui presenti nella base di conoscenza. Siano $C_1, C_2 \in BC_c$ e sia D $\in BC_c \cup \bot$ ogni GCI deve rispettare una delle seguenti forme:

1. $C_1 \sqsubseteq D$
2. $C_1 and C_2 \sqsubseteq D$
3. $C_1 \sqsubseteq \exists R.C_2$
4. $\exists R.C_2 \sqsubseteq D$

Una volta effettuata la normalizzazione della base di conoscenza di partenza, si effettua una query del tipo $C \sqsubseteq D$, si introducono due variabili temporanee X e Y, creando due sussunzioni del tipo $X \sqsubseteq C$ e $D \sqsubseteq Y$, le si normalizza seguendo le forme descritte in precedenza e si aggiungono gli assiomi normalizzati all'interno della base di conoscenza di partenza.

Una volta effettuata la normalizzazione, si può passare a costruire i mapping S ed R, dove:

1. S va da BC_c ad un sottoinsieme di $BC_c \cup T, \bot$
2. R va da R_c a BC_c X BC_c

Dove R_c C denota l'insieme di tutti i nomi di ruolo presenti nella base di conoscenza.

I mapping S ed R vengono inizializzati nel seguente modo:

1. S(C) := C, T per ogni $C \in BC_c$
2. R(r) := $\emptyset$ per ogni $r \in R_c$.

I mapping vengono costruiti seguendo le regole di completamento CR1-CR6, indicate di seguito:

CR1 If $C' \in S(C), C' \sqsubseteq D \in \mathcal{C}$, and $D \notin S(C)$
then $S(C) := S(C) \cup \{D\}$

CR2 If $C_1, C_2 \in S(C), C_1 \sqcap C_2 \sqsubseteq D \in \mathcal{C}$, and $D \notin S(C)$
then $S(C) := S(C) \cup \{D\}$

CR3 If $C' \in S(C), C' \sqsubseteq \exists r.D \in \mathcal{C}$, and $(C, D) \notin R(r)$
then $R(r) := R(r) \cup \{(C, D)\}$

CR4 If $(C, D) \in R(r), D' \in S(D), \exists r.D' \sqsubseteq E \in \mathcal{C}$,
and $E \notin S(C)$
then $S(C) := S(C) \cup \{E\}$

CR5 If $(C, D) \in R(r), \bot \in S(D)$, and $\bot \notin S(C)$,
then $S(C) := S(C) \cup \{\bot\}$

CR6 If $\{a\} \in S(C) \cap S(D), C \rightsquigarrow_R D$, and $S(D) \not\subseteq S(C)$
then $S(C) := S(C) \cup S(D)$

Figura 1.2: Regole CR1 - CR6

Infine, viene verificato se in S(X) è presente Y e in caso affermativo si risponde "si", altrimenti "no". Questo viene fatto grazie al fatto che $C \sqsubseteq D \Leftrightarrow X \sqsubseteq Y$.

3 Implementazione del Reasoner

Nel seguente capitolo verrà fornita una descrizione dettagliata relativa all'implementazione del Reasoner, inclusi i metodi utilizzati. Verranno illustrate le componenti principali della classe, le strutture dati impiegate e le funzionalità offerte per eseguire query sull'ontologia fornita in input. L'implementazione mira a sfruttare e manipolare l'ontologia attraverso vari processi di normalizzazione e gestione degli assiomi, fornendo un sistema efficiente e versatile per l'inferenza delle informazioni.

3.1 Reasoner

La classe Reasoner è progettata per elaborare e inferire informazioni da una data ontologia. La classe viene inizializzata con un'istanza di ontologia e fornisce un metodo pubblico per eseguire query su di essa. La sua implementazione include i seguenti componenti chiave:

- **Costruttore**: Inizializza il reasoner con un'ontologia, prepara il gestore di ontologia, un data factory, e normalizza gli assiomi di sottoclasse dall'ontologia.
- **Metodi e Proprietà**:
  - **Ontology**: L'ontologia fornita in input.
  - **OWLOntologyManager**: Un gestore per creare e manipolare ontologie.
  - **OWLDataFactory**: Una factory per creare entità OWL.
  - **SubClassOfAxioms**: Un insieme di assiomi di sottoclasse estratti dall'ontologia.
  - **NormalizedAxiomsSet**: Un insieme di assiomi normalizzati derivati dall'ontologia di input.
  - **S**: Una mappa per immagazzinare informazioni specifiche derivate dall'ontologia.
  - **R**: Un'altra mappa per memorizzare ulteriori dati rilevanti.

Il metodo pubblico della classe consente di effettuare query sull'ontologia, sfruttando le strutture dati e le trasformazioni preparate durante l'inizializzazione.

3.2 Ontologia

Un'ontologia di test è stata creata utilizzando il software Protegè e si trova nella cartella "data" del progetto. Questa ontologia include:

1. **Classi**: Rappresentate dalle lettere maiuscole dell'alfabeto, dalla A alla V.
2. **Cinque relazioni**: Denominate da r1 a r5.
3. **Due individui**: Identificati come x e y.

Queste classi, relazioni e individui sono organizzati in una struttura di sussunzione, creando un'ontologia composta esclusivamente da assiomi di sottoclasse (SubClassOf).

3.3 Normalizzazione

Il metodo relativo alla normalizzazione prende in input un insieme di OWLAxiom e restituisce un insieme contenente solo relazioni di sottoclasse normalizzate. Queste relazioni di sottoclasse sono standardizzate in quattro forme specifiche, ossia le seguenti:

- $C_1 \sqsubseteq D$
- $C_1 \sqcap C_2$
- $C_1 \sqsubseteq \exists R.C_2$
- $\exists R.C_1 \sqsubseteq D$

Per effettuare la normalizzazione, il metodo esegue un ciclo che attraversa ciascun elemento del set dato in input. Per ogni elemento, vengono eseguiti i seguenti passaggi:

1. **Recupero delle Parti dell'Inclusione**: viene estratta la parte sinistra dell'inclusione e la parte destra dell'inclusione.
2. **Normalizzazione delle Parti**: la parte sinistra viene passata al metodo subClassNormalization() e la parte destra viene passata al metodo superClassNormalization().
3. **Creazione della Relazione di Sottoclasse**: utilizzando il risultato dei metodi subClassNormalization() e superClassNormalization(), viene creata la relazione di sottoclasse finale.

Il risultato di questo processo è quindi un insieme di assiomi di sottoclasse normalizzati, che rispettano le quattro forme previste.

3.4 Metodo subClassNormalization

Il metodo subClassNormalization() prende in input un elemento di tipo OWLClassExpression e restituisce una coppia formata da un insieme e da un OWLClassExpression. L'insieme conterrà tutte le inclusioni necessarie per la corretta normalizzazione dell'input, mentre il secondo elemento della coppia può essere qualsiasi elemento ammesso come parte sinistra di un'inclusione.

OWLClassExpression è una superclasse, quindi il metodo verifica il tipo effettivo dell'espressione, che può essere uno dei seguenti:

1. **OWL_CLASS o OBJECT_ONE_OF**: in questo caso, il metodo ritorna un nuovo oggetto di tipo Pair, con un insieme vuoto come primo elemento e lo stesso OWLClassExpression dato in input come secondo elemento.
2. **OBJECT_INTERSECTION_OF**: Quando l'espressione è un'intersezione, viene chiamato il metodo normalizeIntersectionOf(). Il ritorno di normalizeIntersectionOf() sarà il ritorno del metodo subClassNormalization().

3. **OBJECT_SOME_VALUES_FROM**: Quando l'espressione è un'espressione esistenziale, viene chiamato il metodo normalizeObjectSomeValueFrom(). Il ritorno di normalizeObjectSomeValueFrom() sarà il ritorno del metodo subClassNormalization().

3.5 Metodo superClassNormalization

Il metodo superClassNormalization() prende in input un elemento di tipo OWLClassExpression e restituisce una coppia (Pair) formata da un insieme di OWLSubClassOfAxiom e da un OWLClassExpression. L'insieme contiene tutte le inclusioni necessarie per la corretta normalizzazione dell'input, mentre il secondo elemento della coppia può essere qualsiasi elemento ammesso come parte destra di un'inclusione.

Per effettuare la normalizzazione, il metodo esegue i seguenti passaggi: Viene creato un insieme vuoto di OWLSubClassOfAxiom e vengono inizializzate due variabili, per poi procedere alla verifica del tipo di OWLClassExpression che può essere:

1. **OWL_CLASS o OBJECT_ONE_OF**: Se l'espressione è una classe semplice, viene ritornato un nuovo oggetto di tipo Pair, con un insieme vuoto come primo elemento e lo stesso OWLClassExpression dato in input come secondo elemento.
2. **OBJECT_INTERSECTION_OF**: Se l'espressione è un'intersezione, viene eseguito il cast a OWLObjectIntersectionOf. Viene chiamato il metodo normalizeIntersectionOf() con l'intersezione come argomento. Il risultato di normalizeIntersectionOf() viene passato a reduceToClass() per ottenere la coppia finale. Le inclusioni restituite da normalizeIntersectionOf() vengono aggiunte all'insieme.
3. **OBJECT_SOME_VALUES_FROM**: Se l'espressione è un'espressione esistenziale, viene eseguito il cast a OWLObjectSomeValuesFrom. Viene chiamato il metodo normalizeObjectSomeValueFrom() per ottenere la coppia finale.

Il risultato di questo processo è una coppia contenente le inclusioni necessarie e l'elemento normalizzato.

3.6 Metodo normalizeIntersectionOf

Il metodo normalizeIntersectionOf è un metodo ricorsivo che prende in input un elemento di tipo OWLObjectIntersectionOf. Il metodo estrae la lista degli operandi e per ciascuno di essi verifica se è un'espressione esistenziale; in tal caso, lo normalizza chiamando il metodo normalizeSomeValuesFromAsClass(). Se non è un'espressione esistenziale, è direttamente una classe. A coppie di due, normalizza le varie relazioni di intersezione, trasformandole in semplici classi, fino ad esaurire gli operandi ed effettuare la nuova chiamata ricorsiva passando come parametro l'insieme delle classi appena create.

Nel complesso, il metodo estrae la lista degli operandi dall'elemento OWLObjectIntersectionOf dato in input e per ogni operando, il metodo verifica se si tratta di

un'espressione esistenziale (OBJECT_SOME_VALUES_FROM). Se questo è il caso, allora viene normalizzata chiamando normalizeSomeValuesFromAsClass(). Mentre se non è un'espressione esistenziale, è considerata direttamente una classe.

Dopodiché gli operandi vengono normalizzati a coppie di due, trasformandoli in semplici classi. Questo processo continua ricorsivamente fino a esaurire gli operandi.

Alla fine il metodo restituisce una coppia (Pair) formata da un insieme di relazioni di inclusione e un elemento di tipo OWLClassExpression che rappresenta l'intersezione tra due classi semplici.

3.7 Metodo normalizeSomeValuesFromAsClass

Il metodo normalizeSomeValuesFromAsClass prende in input un elemento di tipo OWLObjectSomeValuesFrom e lo passa come parametro al metodo normalizeObjectSomeValueFrom(). Questo metodo restituisce un'espressione esistenziale, che viene successivamente trasformata in una singola classe. Tale trasformazione è utile quando, ad esempio, un'espressione esistenziale è presente come operando di un'intersezione e deve essere ridotta a una singola classe.

Quindi l'elemento OWLObjectSomeValuesFrom dato in input viene passato al metodo normalizeObjectSomeValueFrom() che restituisce un'espressione esistenziale normalizzata. Questa espressione esistenziale viene poi trasformata in una singola classe.

Al termine, il metodo restituisce una coppia (Pair) formata da insieme di Relazioni di Inclusione, cioè un insieme contenente tutte le relazioni di inclusione necessarie per la normalizzazione e un elemento di tipo OWLClassExpression che rappresenta una singola classe semplice risultante dalla normalizzazione dell'espressione esistenziale.

3.8 Metodo reduceToClass

Il metodo reduceToClass prende in input un'espressione di tipo OWLClassExpression, verifica se tale parametro corrisponde a un'intersezione o a un'espressione esistenziale, e lo riduce opportunamente a una nuova classe creata tramite il metodo createTempClass().

Controlla se l'espressione è un'intersezione o un'espressione esistenziale e riduce l'espressione a una nuova classe utilizzando createTempClass().

Il metodo restituisce una coppia (Pair) formata da un insieme contenente le relazioni di inclusione risultanti dalla riduzione ed un elemento di tipo OWLClassExpression che rappresenta la nuova classe creata.

3.9 Metodo createTempClass

Il metodo createTempClass è responsabile di creare un nuovo elemento di tipo OWLClass, denominato ad esempio TEMPx, dove x è un numero intero che viene incrementato di volta in volta. Il metodo restituisce in output l'elemento di tipo OWLClass appena creato.

3.10 Metodo normalizeSingleIntersectionOf

Il metodo normalizeSingleIntersectionOf prende in input tre parametri: due parametri di tipo OWLClassExpression, che rappresentano le due classi che formano un'intersezione ed un parametro di tipo OWLClass, che rappresenta una nuova classe precedentemente creata e che alla fine del metodo diventerà equivalente all'intersezione originaria.

Il metodo crea le relazioni di inclusione necessarie per stabilire che la nuova classe è equivalente all'intersezione delle due classi originali ed infine ritorna un insieme di relazioni di inclusione, che documentano come la nuova classe è costruita a partire dall'intersezione delle due classi. Quindi, essenzialmente il metodo ritorna un insieme di relazioni di inclusione che descrivono la costruzione della nuova classe equivalente all'intersezione delle due classi date in input.

3.11 Metodo normalizeSingleObjectSomeValuesFrom

Il metodo normalizeSingleObjectSomeValuesFrom prende in input due parametri: un parametro di tipo OWLObjectSomeValuesFrom, che rappresenta un'espressione esistenziale ed un parametro di tipo OWLClass, che rappresenta una nuova classe precedentemente creata e che alla fine del metodo diventerà equivalente all'esistenziale originario.

Il metodo crea le relazioni di inclusione necessarie per stabilire che la nuova classe è equivalente all'esistenziale originario e ritorna un insieme di relazioni di inclusione, che documentano come la nuova classe è costruita a partire dall'esistenziale dato in input. Quindi, in questo caso il metodo ritorna un insieme di relazioni di inclusione che descrivono la costruzione della nuova classe equivalente all'esistenziale dato in input.

3.12 Metodo doQuery

Il metodo doQuery è l'unico metodo pubblico della classe e consente agli utenti di sottoporre una query al reasoner. Prende in input un parametro di tipo OWLSubClassOfAxiom, estrae la parte sinistra e destra dell'inclusione e le passa come parametri al metodo createFictitious. Il risultato di questo metodo viene successivamente normalizzato.

Il metodo:

- Estrae la parte sinistra e destra dell'OWLSubClassOfAxiom.

- Le parti sinistra e destra dell'inclusione vengono passate come parametri al metodo createFictitious. Questo metodo prende in input due parametri di tipo OWLClassExpression, che rappresentano le due parti di un'inclusione. Il metodo si occupa di creare due nuove classi, denominate X e Y, e di stabilire le relazioni di sottoclasse tra queste nuove classi e le parti dell'inclusione e quindi ritorna un insieme che contiene le relazioni di sottoclasse create.
- Il risultato ottenuto da createFictitious viene normalizzato.
- L'ontologia normalizzata e il risultato ottenuto vengono combinati per formare un nuovo set.
- Questo nuovo set viene passato come parametro alla funzione initializeMapping e successivamente alla funzione applyingCompletionRules.
- Il metodo effettua un controllo per verificare se i due elementi creati da createFictitious risultano essere uno incluso nell'altro, restituendo un valore booleano che indica il risultato del controllo.

3.13 Metodo initializeMapping

Il metodo initializeMapping si occupa di inizializzare le mappe R ed S, che sono attributi della classe, utilizzando gli assiomi di sottoclasse forniti come input. Prende quindi in input un insieme contenente solo elementi di tipo OWLSubClassOfAxiom. Da ciascuno di questi assiomi, vengono estratte le due parti dell'inclusione e ognuna di esse viene passata come parametro al metodo initializeSingleMapping.

Dal set di assiomi di tipo OWLSubClassOfAxiom, vengono recuperate le due parti dell'inclusione e per ciascuna coppia di parti dell'inclusione estratte e le parti dell'inclusione vengono passate come parametri al metodo initializeSingleMapping.

Il metodo non ritorna nulla in quanto le mappe R ed S sono istanziate come attributi di classe.

3.14 Metodo initializeSingleMapping

Il metodo initializeSingleMapping è responsabile di inizializzare le mappe R e S della classe e prende in input un parametro di tipo OWLClassExpression, che può essere di diversi tipi: OWL_CLASS oppure OBJECT_ONE_OF, OBJECT_INTERSECTION_OF, OBJECT_SOME_VALUES_FROM. In ciascuno di questi casi, vengono eseguite le operazioni appropriate per inizializzare le mappe R e S.

Il metodo essenzialmente controlla il tipo dell'OWLClassExpression fornito come input per determinare il tipo di operazioni da eseguire e non ritorna nulla in quanto le mappe R ed S sono istanziate come attributi di classe e vengono modificate direttamente all'interno del metodo.

3.15 Metodo applyingCompletionRules

Il metodo applyingCompletionRules prende in input un insieme contenente solo elementi di tipo OWLSubClassOfAxiom. Questo metodo implementa una serie di cicli for per applicare correttamente le "Completion Rules" - CR1, CR2, CR3, CR4, CR5, CR6.

Nel dettaglio: il metodo itera attraverso l'insieme di assiomi di sottoclasse forniti come input e durante ciascuna iterazione, vengono applicate le regole di completamento (CR1, CR2, CR3, CR4, CR5, CR6) ai dati contenuti nelle mappe R ed S, che sono istanziate come attributi di classe.

Le regole vengono applicate per garantire che le informazioni all'interno delle mappe R ed S siano correttamente integrate e che l'ontologia sia coerente secondo le specifiche delle regole di completamento.

Inoltre il metodo modifica direttamente le mappe R ed S all'interno della classe, senza ritornare alcun valore.

3.16 Metodi CR1 - CR6

I metodi relativi alle regole di completamento sono i seguenti:

I metodi da CR1-CR4 implementano una logica con cui ognuno di essi prende in input un OWLClassExpression e un insieme di OWLSubClassOfAxiom. Il compito è verificare se la mappa S, relativa alla chiave specificata, viene modificata secondo le specifiche definite dalla regole CR1 - CR4.

CR1 If $C' \in S(C), C' \sqsubseteq D \in \mathcal{C}$, and $D \notin S(C)$
then $S(C) := S(C) \cup \{D\}$

CR2 If $C_1, C_2 \in S(C), C_1 \sqcap C_2 \sqsubseteq D \in \mathcal{C}$, and $D \notin S(C)$
then $S(C) := S(C) \cup \{D\}$

CR3 If $C' \in S(C), C' \sqsubseteq \exists r.D \in \mathcal{C}$, and $(C, D) \notin R(r)$
then $R(r) := R(r) \cup \{(C, D)\}$

CR4 If $(C, D) \in R(r), D' \in S(D), \exists r.D' \sqsubseteq E \in \mathcal{C}$,
and $E \notin S(C)$
then $S(C) := S(C) \cup \{E\}$

Figura 1.3: Regole CR1 - CR4

3.17 Metodo CR5

Il metodo CR5 prende in input un parametro di tipo OWLClassExpression e verifica se la mappa S, relativa a questa chiave, viene modificata secondo le specifiche definite dalla regola di completamento CR5.

CR5 If $(C, D) \in R(r)$, $\bot \in S(D)$, and $\bot \notin S(C)$,
then $S(C) := S(C) \cup \{\bot\}$

Figura 1.4: Regola CR5

3.18 Metodo CR6

Il metodo CR6 prende in input tre parametri: due di tipo OWLClassExpression e uno di tipo DefaultDirectedGraph. Il suo compito è verificare se esiste un cammino nel grafo diretto fornito come parametro, dal nodo rappresentato da key1 al nodo rappresentato da key2. Se tale cammino esiste, il metodo modifica la mappa S associando key1 a key2 secondo le specifiche indicate nella figura CR6.

Nel dettaglio utilizza un grafo diretto dove:

- I nodi sono rappresentati da tutte le classi presenti nell'ontologia su cui si sta lavorando.
- Gli archi sono rappresentati dalle coppie presenti nella mappa R.
- Il metodo CR6 verifica se esiste un cammino nel grafo diretto da key1 a key2. Se esiste un cammino nel grafo da key1 a key2, allora la mappa S viene modificata associando key1 a key2.

CR6 If $\{a\} \in S(C) \cap S(D)$, $C \rightsquigarrow_R D$, and $S(D) \not\subseteq S(C)$
then $S(C) := S(C) \cup S(D)$

Figura 1.5: Regola CR6

3.19 Metodo checkBottom

Il metodo checkBottom prende in input un parametro di tipo OWLClassExpression e si occupa di verificare se tra le classi incluse nell'espressione passata sia presente il concetto "bottom". Se il concetto "bottom" è presente, il metodo solleva un'eccezione di tipo IllegalArgumentException. Se il concetto "bottom" non è presente, il metodo termina senza fare nulla.

3.20 Metodo subAndSuperCheckBottom

Il metodo subAndSuperCheckBottom prende in input due parametri di tipo OWLClassExpression e si occupa di chiamare il metodo checkBottom su entrambi i parametri. Questo permette di verificare se ciascuna delle due espressioni contiene il concetto "bottom". Se una delle due espressioni contiene il bottom in una posizione non consentita, il metodo checkBottom solleverà un'eccezione di tipo IllegalArgumentException.

4 Testing del Reasoner

Per verificare il corretto funzionamento del reasoner EL++, è stato condotto un processo di testing utilizzando una varietà di query. L'obiettivo principale di questi test era assicurarsi che il reasoner fosse in grado di determinare correttamente le relazioni di sussunzione tra i concetti.

In particolare, sono state formulate e somministrate due tipologie principali di query:

1. **Query di Sussunzione (Vere)**: create in modo che la relazione di sussunzione tra i concetti fosse presente. Il reasoner in questo caso deve essere in grado di riconoscere correttamente questa relazione e restituire un risultato positivo, confermando la sussunzione.

2. **Query di Sussunzione (False)**: cioè query in cui la relazione di sussunzione non sussiste. L'obiettivo è stato quello di verificare che il reasoner fosse in grado di identificare correttamente l'assenza di tale relazione e restituire un risultato negativo.

L'analisi dei risultati ottenuti ha dimostrato che il reasoner EL++ è stato in grado di rispondere accuratamente a entrambe le tipologie di query.

Prima di descrivere in dettaglio i test che sono stati effettuati, occorre presentare l'ontologia realizzata tramite l'ambiente Protegè. In particolare, è stata utilizzata la sezione "OntoGraf" di Protegé la quale consente di osservare una rappresentazione grafica dell'ontologia (mostrata nella figura seguente).

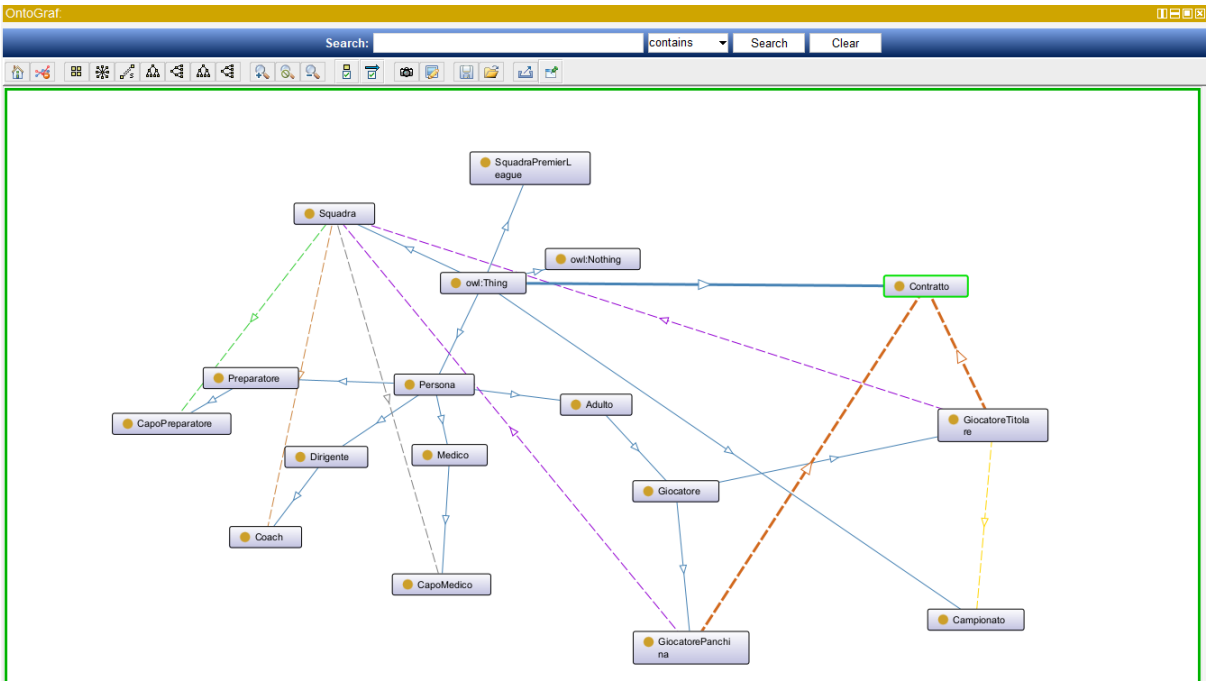


Figura 4.1: Ontologia

Per ogni test eseguito sul Reasoner, è stato possibile confrontare l'esito di tale test attraverso il Reasoner Hermit presente all'interno di Protegé. In particolare, i test eseguiti sono elencati di seguito:

1.
   - **Query**: *GiocatoreTitolare* and *GiocatorePanchina* $\sqsubseteq$ *Giocatore*
   - **Esito**: True
   - **Hermit da Protegé**: True
2.
   - **Query**: *GiocatoreTitolare* and *GiocatorePanchina* $\sqsubseteq T$
   - **Esito**: True
   - **Hermit da Protegé**: True
3.
   - **Query**: $T \sqsubseteq$ *GiocatoreTitolare* and *GiocatorePanchina*
   - **Esito**: False
   - **Hermit da Protegé**: False
4.
   - **Query**: *Preparatore* and *Medico* and *Dirigente* and *Adulto* $\sqsubseteq$ *Persona*
   - **Esito**: True
   - **Hermit da Protegé**: True
5.
   - **Query**: *Coach* $\sqsubseteq \exists$*iscrittoA*.*PremierLeague*
   - **Esito**: False
   - **Hermit da Protegé**: False
6.
   - **Query**: *GiocatorePanchina* $\sqsubseteq \exists$*haContratto*.*Contratto*
   - **Esito**: True
   - **Hermit da Protegé**: True
7.
   - **Query**: *GiocatorePanchina* $\sqsubseteq \exists$*haSquadra*.*Dirigente*
   - **Esito**: False
   - **Hermit da Protegé**: False
8.
   - **Query**: *GiocatorePanchina* $\sqsubseteq \exists$*haCapoMedico*.*Medico* and $\exists$*haCoach*.*Coach* and $\exists$*haCapoPreparatore*.*Preparatore*
   - **Esito**: True
   - **Hermit da Protegé**: True

Qualche esempio di query lanciate su Protegè per il reasoner Hermit (versione 1.4.3.456) sono date dalle seguenti immagini:

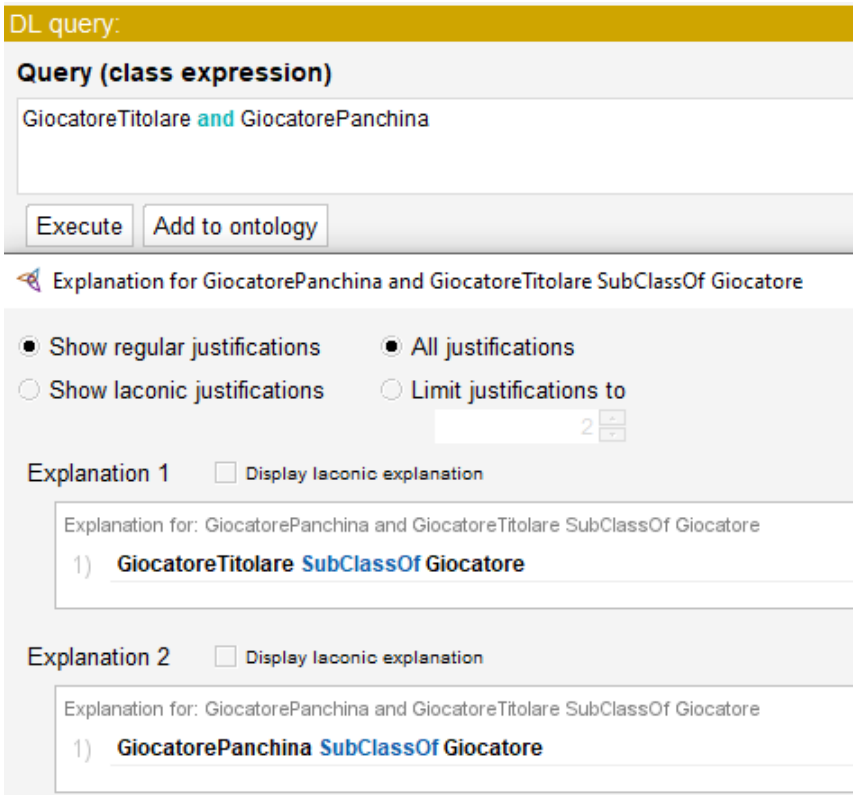


Figura 4.2: Query 1 - Hermit

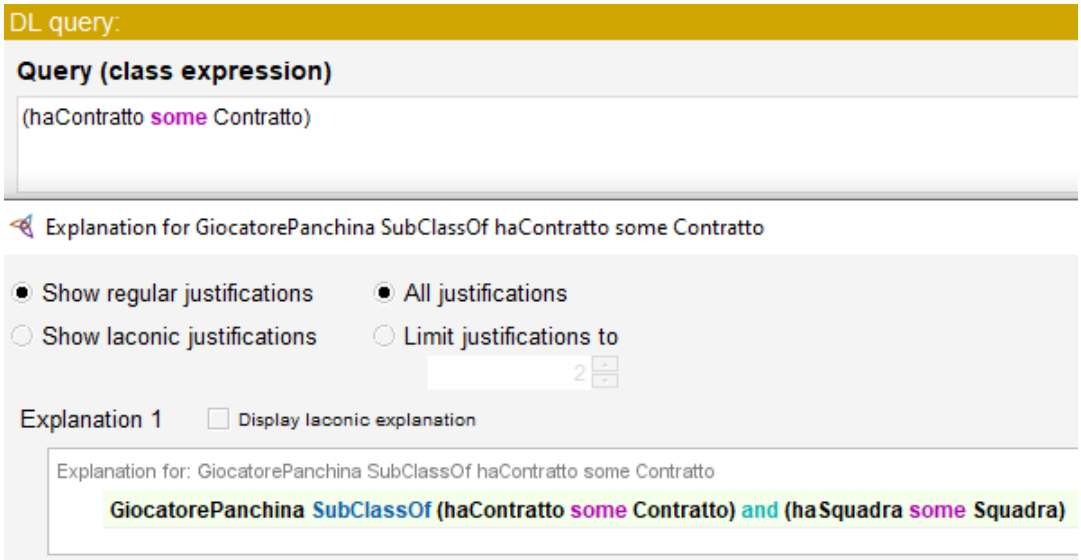


Figura 4.2: Query 6 - Hermit