Motore inferenziale per la logica ALC

Alessandro Mauro

A.A. 2021/2022

Prefazione

In questo documento è descritta la realizzazione di un motore inferenziale per la logica ALC mediante la tecnica dei tableaux. In particolare, tale ragionatore esegue le seguenti operazioni:

- Decide il problema della (in)soddisfacibilità di un concetto C rispetto ad una TBox T. La TBox è non vuota.
- Il tableaux sfrutta la tecnica del ***lazy unfolding*** per velocizzare la computazione.
- È utilizzata la tecnica del ***blocking***.
- Il tableaux risultante da una interrogazione è esportato in formato *rdf* e visualizzato mediante un programma di graph visualization.

In questo documento sono esplicate le tecniche e le metodologie per lo sviluppo del motore inferenziale. Il linguaggio di programmazione utilizzato è ***Java***. Sono utilizzate le seguenti API per poter implementare il motore inferenziale:

- **OWL API**: Permette di creare, manipolare e gestire le ontologie e i concetti OWL.
- **Jena RDF API**: È un API Java per le applicazioni di semantic web. Il pacchetto contiene interfacce per la rappresentazione di modelli, risorse, proprietà, litterali e tutti gli altri concetti chiave di RDF.
- **graphviz-java**: Graphviz è un software per la creazione e visualizzazione di grafi. graphviz-java è un API che permette di utilizzare **Graphviz** tramite Java. Crea dei modelli graphviz e permette di convertirli in diversi formati (come PNG o SVG).

Inoltre è utilizzato il framework **Swing** per la creazione di una basilare interfaccia grafica che permette l'acquisizione degli input (ontologia e concetto) e la visualizzazione dell'output (grafo e file RDF).

Tutte le dipendenze delle API sono state inserite nel file *pom.xml*, generato alla creazione del progetto.

Flusso del programma

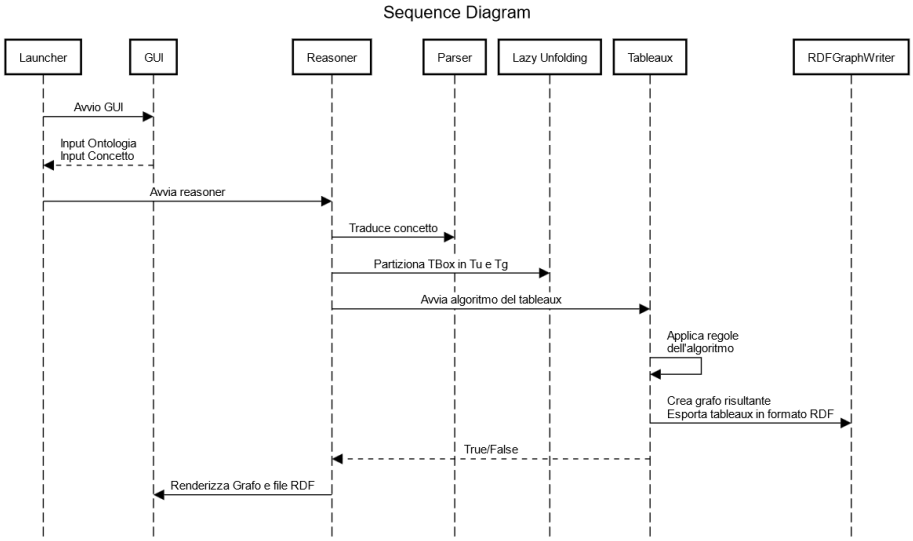

Figure 1: Diagramma di sequenza dell'applicazione

Il diagramma in figura 1 mostra il flusso dell'esecuzione del programma. In tale diagramma non sono citate alcune classi ausiliare allo sviluppo del programma, come la classe ***Node*** (che contiene le informazioni per memorizzare un nodo inteso come individuo OWL) e la classe ***Ontology*** (che crea l'ontologia a partire da un file generando la corrispettiva TBox).

Ogni momento cruciale del flusso dell'esecuzione è esplicato nella seguente lista:

1. **Avvio GUI**:

   Il flusso d'esecuzione inizia dal *Launcher*. Il Launcher è la classe contiene il *main* e permette l'avvio del programma. All'avvio dell'esecuzione, è possibile selezionare gli input (ontologia e concetto) attraverso degli appositi pannelli. La GUI ritorna il nome del file in cui è presente l'ontologia e il concetto in formato Stringa. La GUI è mostrata nella figura 2

2. **Avvio Reasoner**:

   Dopo aver acquisito gli input, il Launcher avvia il reasoning chiamando il metodo apposito del Reasoner. Il Reasoner è la classe che prepara i dati per poter poi eseguire l'algoritmo dei tableaux. In particolare:

   - Traduce il concetto da stringa in OWLClassExpression tramite la classe ***Parser***.

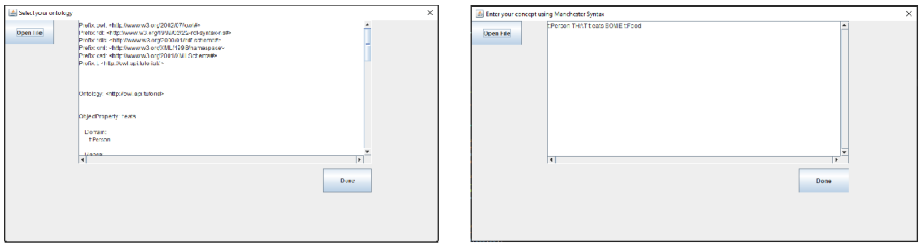

Figure 2: A sinistra il pannello per acquisire l'ontologia. A destra il pannello per acquisire il concetto nel formato di Manchester

- Partiziona la TBox in T_u e T_g per poter applicare il lazy unfolding (T_u contenente solo assiomi unfoldable, T_g contenente la restante parte di assiomi) tramite la classe ***Lazy Unfodling***.
- Traduce la T_g in un concetto C' tramite il Parser. Senza lazy unfolding si sarebbe dovuta tradurre l'intera TBox.
- Crea il primo nodo il quale deve soddisfare il concetto C e la T_g C'.
- Crea il tableaux e chiama l'algoritmo del tabelaux sul nodo starter.

3. **Traduzione concetto**:

   Il concetto acquisito tramite la GUI è di tipo stringa. È necessario, dunque, convertirlo in tipo *OWLClassExpression*. La classe Parser ha un metodo apposito per fare ciò. Converte la stringa in un concetto OWL attraverso il parser *ManchesterOWLSyntaxParser*. Utilizzando questo parser, è anche possibile interpretare i prefissi, evitando così di scrivere l'intero URI nel pannello di acquisizione.

4. **Partizionamento TBox in T_u e T_g**:

   Il reasoner chiama il metodo di partizionamento della TBox della classe *Lazy Unfolding*. Il metodo partiziona la TBox creando due liste di assiomi: T_u e T_g. T_u contiene solo assiomi *unfoldable* (ossia del tipo A $\sqsubseteq$ D, oppure A $\equiv$ D tale che se A $\equiv$ è in T_u, allora A non compare a sinistra di nessun'altra CGI di T_u), mentre T_g contenente la restante parte di assiomi.

   Dopo aver partizionato la TBox in T_u e T_g, T_g viene tradotta in un singolo concetto (in quanto stiamo operando con una TBox non vuota). Dunque, si crea il primo nodo che deve soddisfare sia il concetto di input, che la traduzione in singolo concetto della T_g. A questo punto è possibile avviare l'algoritmo del tableaux.

5. **Avvio algoritmo del tableaux**:

   L'algoritmo del tableaux è una tecnica di ragionamento che risolve il problema della (in)soddisfacibilità di un concetto C. Tenta di trovare un mod-

ello che lo soddisfi: Se lo trova, allora C è soddisfacibile. Se non lo trova, allora C è insoddisfacibile.

Siccome si sta operando con una TBox non vuota, l'algoritmo del tableaux fa uso della tecnica di **blocking**, che consente di bloccare un nodo se: $L(x_j) \subseteq L(x_i)$ e $i < j$. In questo caso x_i blocca x_j.

Oltre alla tecnica del blocking, l'algoritmo del tableaux fa uso della tecnica di ottimizzazione **Lazy Unfolding**, che consente di *aggiungere* le definizioni dei *concept name* alla corrente Label che il nodo deve soddisfare.

6. **Applicazione regole dell'algoritmo del tableaux**:

   L'algoritmo del tableaux esegue le seguenti operazioni:

   (a) Applica AND esaustivamente.

   (b) Applica OR esaustivamente.

   (c) Applica la regola di lazy unfolding.

   (d) Applica la regola dell'Esistenziale.

   (e) Applica la regola dell'Universale.

   Dopo l'applicazione di tali regole, l'algoritmo ritornerà se sono avvenuti eventuali clash, oppure se il concetto è soddisfacibile.

7. **Esportazione del grafo e del file RDF**

   Una volta terminato l'algoritmo del tableaux, esso genera sia un grafo (tramite Graphviz), sia un file contenente il tableaux in formato *RDF*. Queste due esportazioni sono salvate in una cartella *results*.

Manuale d'uso

Scelta ontologia

Appena avviato il programma, compare a schermo un pannello in cui è possibile **selezionare da file** (tramite il pulsante ***Open File***) **l'ontologia** desiderata. Dopo aver selezionato l'ontologia, premere il pulsante ***Done*** per andare avanti.

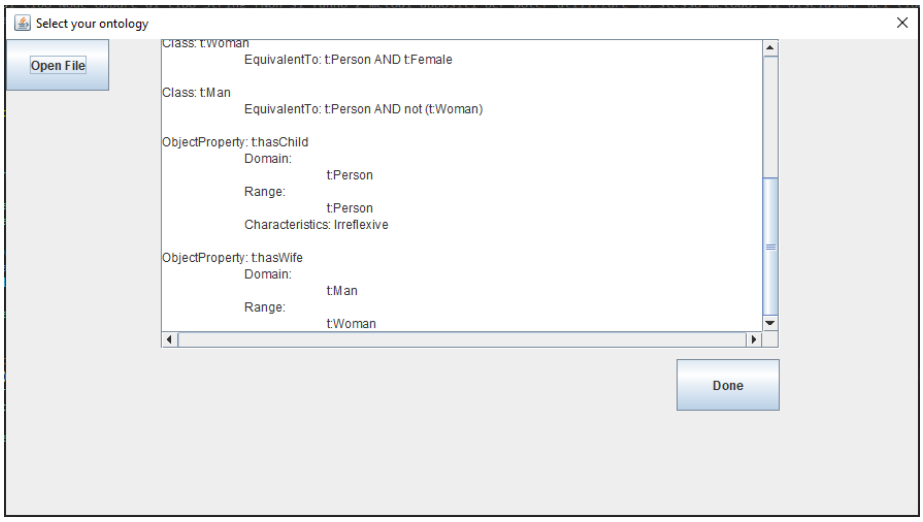

Figure 3: Scelta dell'ontologia

Scelta concetto

Dopo aver selezionato l'ontologia, compare a schermo un pannello in cui è possibile **selezionare il concetto** di cui si vuole stabilire la (in)soddisfacibilità. Il concetto può essere sia selezionato da file (tramite il pulsante ***Open File***), sia essere scritto nell'apposito campo di testo. In entrambi i casi, il concetto deve rispettare la **sintassi di Manchester**. Dopo aver selezionato (o scritto) il concetto, premere il pulsante ***Done*** per andare avanti.

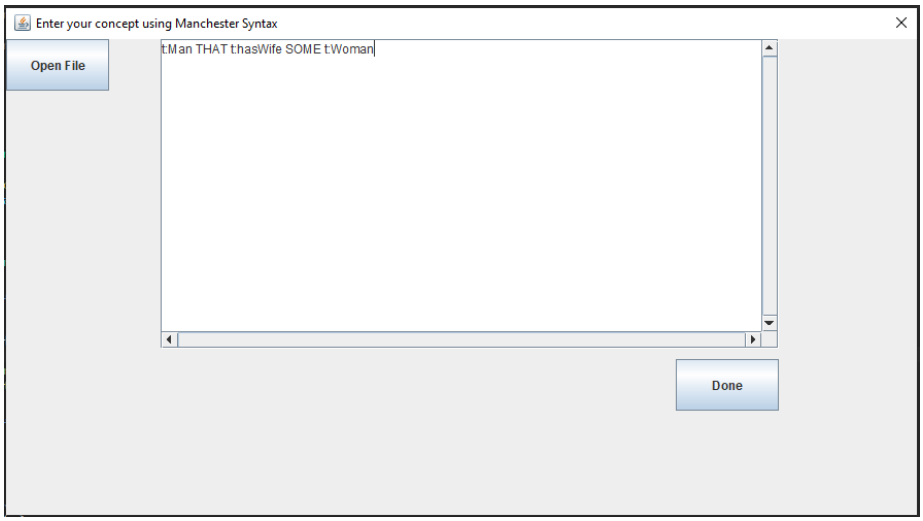

Figure 4: Scrittura o selezione da file del concetto

Risultati

Dopo un breve lasso di tempo dall'acquisizione degli input, a schermo compare un pannello che mostra i **risultati** dell'esecuzione. I risultati comprendono la **(in)soddisfacibilità del concetto**, il **tempo di esecuzione** e la possibilità di **visualizzare sia il grafo risultante, sia il file RDF**.

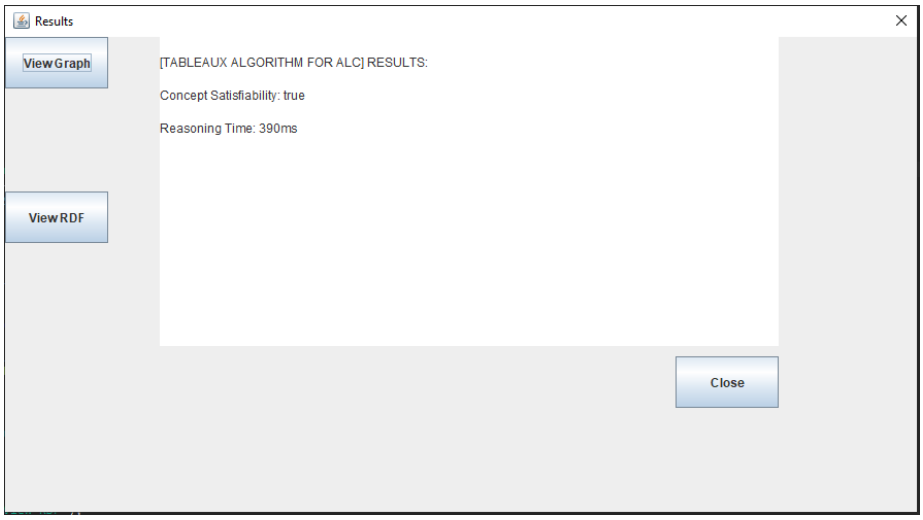

Figure 5: Pannello comprendente i risultati dell'esecuzione

Visualizzazione grafo

Cliccando su ***View Graph*** dal pannello *Risultati* (fig. 5), si apre il grafo risultante dall'algoritmo del tableaux in formato *SVG*. Nel grafo sono presenti i vari nodi che sono stati generati dall'esecuzione. Ogni nodo è legato ad una nota che contiene tutti i concetti che il nodo deve soddisfare. Gli archi tra un nodo e l'altro sono labellati con la regola che ha causato la generazione del nodo (come l'applicazione della regola OR). Gli archi loop sui vari nodi indicano le operazioni che sono state fatte sullo stesso nodo che non hanno causato la generazione di nuovi figli (come applicazione regola AND, applicazione Lazy Unfolding). L'applicazione della regola Lazy Unfolding genera una nuova nota contenente i concetti che il nodo deve soddisfare a cui sono aggiunte le definizioni dei concept name (mostrato in figura 6).

I nodi che hanno generato un ***clash*** sono identificati con un cerchio rosso con un collegamento ad una nota che indica i concetti che hanno generato il *clash*. I nodi ***clash-free*** sono cerchiati di verde con un collegamento ad un rettangolo che indica che il nodo è *clash-free* (figura 7). Ovviamente, è sufficiente che almeno un nodo sia *clash-free* affinché il risultato della computazione sia *true*.

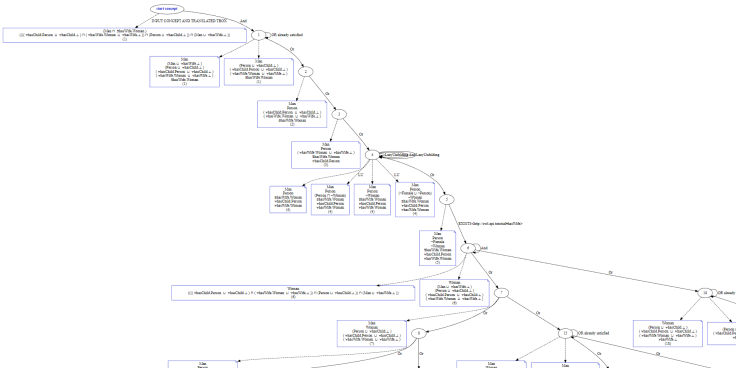

Figure 6: Grafo risultante

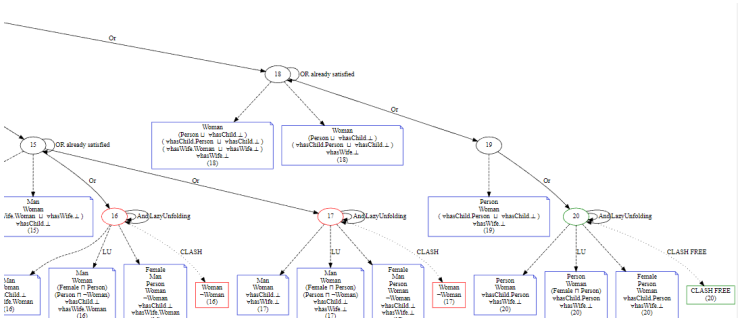

Figure 7: Clash e clash free

Visualizzazione RDF

Cliccando sul pulsante ***View RDF*** dal pannello *Risultati* (fig. 5), si apre un file contenente le triple RDF scritte nel formato ***Turtle***. Ogni nodo contiene la *property* ***labels*** che indica tutti i concetti che il nodo deve soddisfare, ed eventualmente le *property* ***orEdge*** oppure ***existentialRule*** seguito dall'*IRI* del nodo che essi generano.

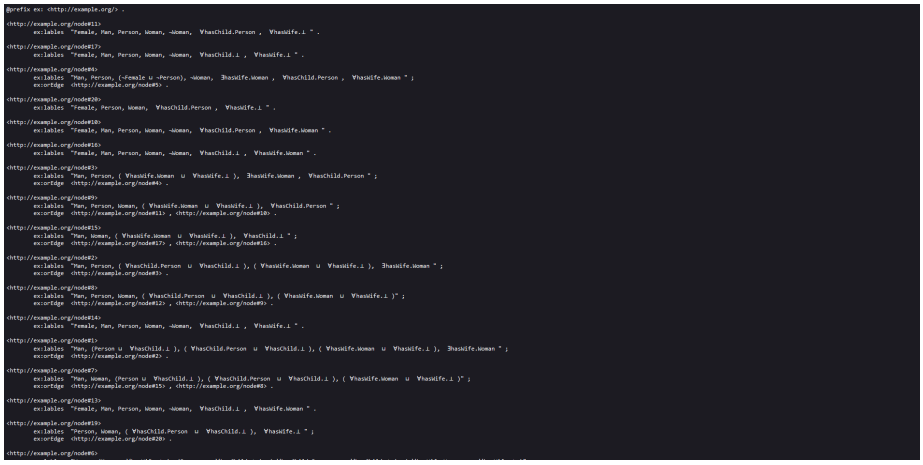

Figure 8: Risultato in formato RDF