Progetto Intelligent Web

### Guida all’uso

## Rauso Giuseppe N97000357 Romano Salvatore N97000372

# Struttura della directory

.

├── demo/

│ ├── pom.xml

│ └── src/

│ └── main/

│ └── java/

│ └── com/

│ └── alcreasoning/

│ ├── App.java

│ ├── GraphDrawer.java

│ ├── OntologyPreprocessor.java

│ ├── Reasoner.java

│ └── visitors/

│ ├── AllVisitors.java

│ ├── AndVisitor.java

│ ├── AtomicConceptVisitor.java

│ ├── LazyUnfoldingVisitor.java

│ ├── OrAndPreprocessorVisitor.java

│ ├── OrVisitor.java

│ └── PrinterVisitor.java

├── graphs/

│ └── // File svg (graphviz) e rdf generati dal programma

├── labels/

│ └── // File contenenti i label dei nodi per i grafi graphviz

├── midgard.owl

├── query\_esempio.txt

└── query.pdf

Per semplicità di visualizzazione sono stati omessi file e directory generati automaticamente da visual studio.

Nel file App.java è possibile trovare il **main** per poter eseguire il programma.

Il file midgard.owl contiene una ontologia di esempio utilizzata per le query riportate in query\_esempio.txt.

# Dipendenze

Il progetto è stato sviluppando utilizzando **Maven** per gestire le librerie. Nel file pom.xml sono presenti le dipendenze. È possibile quindi configurare un progetto Maven e utilizzare questo file per installare automaticamente le librerie utilizzate. In particolare, le dipendenze principali sono le seguenti:

* OWL API

<dependency>

      <groupId>net.sourceforge.owlapi</groupId>

      <artifactId>owlapi-distribution</artifactId>

      <version>5.1.18</version>

</dependency>

* Graphviz (guru.nidi)

<dependency>

      <groupId>guru.nidi</groupId>

      <artifactId>graphviz-java</artifactId>

      <version>0.18.1</version>

</dependency>

* Jena

<dependency>

      <groupId>org.apache.jena</groupId>

      <artifactId>apache-jena-libs</artifactId>

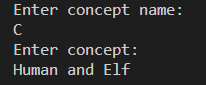
      <type>pom</type>

      <version>3.17.0</version>

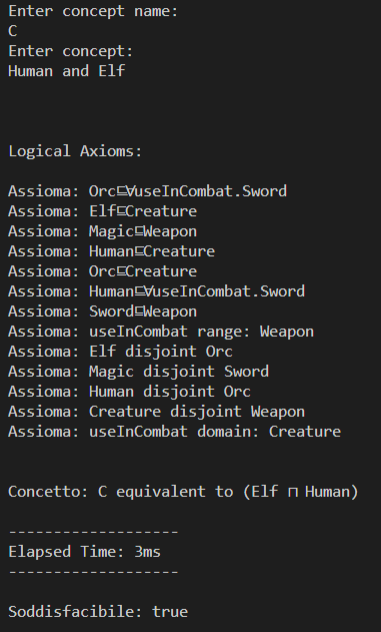
</dependency>

# Utilizzo del programma

Eseguendo il main, nel terminale verrà richiesto di inserire prima il nome della query e poi il concetto nel formato *Manchester*. Si avrà quindi un input di questo tipo:

**Esempio**

Per la query va inserito prima C come concept name e poi Human and Elf come concept.

In output sono riportati gli assiomi logici presenti nella KB, il concetto inserito, il tempo impiegato per l’esecuzione dell’algoritmo del tableau e il risultato dell’algoritmo.

# Lazy unfolding e grafo in output

É possibile eseguire l’algoritmo con il **lazy unfolding** e generare un grafo in output modificando i parametri della funzione run\_tableau:

static String run\_tableau(boolean lazy\_unfolding, <- true per lazy unfolding

boolean draw\_graph, <- true per grafo

String save\_path,

OntologyPreprocessor preproc)

# Ontologia midgard.owl

