

# **SYMPTOM ANALYZER: Documentazione**

**A.A 2024/2025**

**Progetto di Ingegneria della conoscenza**

Progetto a cura di:

- Appice Lorenzo, 777747 : [l.appice@studenti.uniba.it](mailto:l.appice@studenti.uniba.it)

Professore del corso: Fanizzi Nicola

Github del progetto: <https://github.com/lorenzoappice/ICON24-25.git>

## **Indice dei contenuti documentazione:**

0. Decisioni di sviluppo;
1. Introduzione;
2. Diagramma delle classi;
3. Struttura interfaccia;
4. Caso d'uso;
5. Architettura;
6. Conclusioni;

## **Elenco argomenti di interesse:**

- **Knowledge Graph e Ontologie;**
- **Grafi orientati** (nodi sono malattie e sintomi)
- **Rappresentazione e Ragionamento Proposizionale** (bottom-up, Spiegazioni a Livello di Conoscenza (how));
- **Stima probabilistica** (della diagnosi);

## 0 Decisioni di sviluppo

L'applicazione in questione è stata sviluppata in python 3.11 e come IDE ho utilizzato "pycharm" con licenza per studenti.

Ecco le librerie presenti nel progetto e scelte per l'utilizzo:

- **tkinter**: è stata usata per creare l'interfaccia grafica (GUI), dove **tk** è il modulo principale, mentre **ttk** (Themed Tkinter) l'ho usato per un widget con un aspetto più moderno, **messagebox** l'ho usato per mostrare finestre di avviso o errore.
- **os**: Importata in **utils.py**, per la gestione di percorsi di file.
- **owlready2**: Usata per gestire ontologie in formato OWL, usata per caricare, manipolare e interrogare ontologie per rappresentare conoscenze strutturate.

### Moduli presenti:

- **Ontologymanager**: Gestisce la logica dell'ontologia, come il caricamento e la mappatura delle malattie con i sintomi e contiene funzioni come **create\_KB()**, **build\_map()**, e **list\_symptoms()**.
- **utility**: Fornisce funzioni di supporto, come **get\_prob\_model()** per calcolare la probabilità di una malattia in base ai sintomi inseriti e contiene **how()**, che restituisce la clausola usata per dimostrare un'affermazione.
- **logicProblem**: Definisce la rappresentazione logica delle clausole e delle basi di conoscenza e contiene classi come **Clause** e **KB** per gestire la logica della base di conoscenza.
- **logicBottomUp**: Implementa un algoritmo bottom-up per l'inferenza logica sulle basi di conoscenza, contiene la funzione **fixed\_point()**, che calcola il punto fisso della KB.
- **interfaccia**: Gestisce l'interfaccia dell'applicazione, è il punto di avvio del programma.

### **Ontologia utilizzata:**

È stato utilizzato un dataset estratto dal sito [Disease Ontology](#).

Il file contiene informazioni strutturate sulle malattie umane, con l'obiettivo di fornire descrizioni coerenti e riutilizzabili per la comunità biomedica.

Le principali entità incluse nel dataset sono:

- **Classi di malattie:** con identificatori univoci e relazioni gerarchiche.
- **Descrizioni e definizioni:** che specificano il significato di ogni malattia.
- **Annotazioni semantiche:** come sinonimi e riferimenti ad altre ontologie biomediche (es. OBO, Gene Ontology).
- **Metadati:** inclusi autori, date di generazione e licenza d'uso.
- **Sintomi associati,** che descrivono le manifestazioni cliniche principali di ciascuna malattia.

Il dataset segue il modello RDF (Resource Description Framework) e utilizza standard come OWL (Web Ontology Language) per garantire interoperabilità e riusabilità dei dati.

# 1 INTRODUZIONE:

**Symptom Analyzer** è un'applicazione innovativa che sfrutta una semplice e intuitiva interfaccia utente per aiutare gli utenti a individuare possibili malattie sulla base dei sintomi che manifestano.

L'obiettivo principale dell'applicazione è quello di fornire un supporto diagnostico utile per chiunque abbia bisogno di una guida preliminare sui possibili disturbi legati ai sintomi avvertiti.

Il cuore di questa applicazione si fonda sull'integrazione di un'ontologia avanzata, la **Disease Ontology (DO)**, un sistema altamente standardizzato progettato per offrire una descrizione precisa e coerente delle malattie umane.

DO è una risorsa fondamentale per la comunità biomedica, in quanto fornisce un vocabolario unificato e ben strutturato per i termini medici legati alle malattie e ai concetti ad esse associati.

Grazie a questa ontologia, il Symptom Analyzer è in grado di rappresentare in modo accurato le relazioni tra i diversi sintomi e le malattie, permettendo di correlare informazioni.

Il modello alla base dell'applicazione è costruito su un **grafo orientato**, dove i nodi del grafo rappresentano due elementi principali: le **malattie** e i **sintomi**. In questo grafo, le **malattie** sono rappresentate da nodi che collegano a loro volta una serie di **sintomi**, creando un sistema di relazioni bidirezionali.

Gli **archi** che escono da un nodo malattia convergono su uno o più nodi sintomo, stabilendo un legame diretto che indica che una determinata malattia è associata ad un sintomo specifico. Al contrario, i sintomi sono collegati alle malattie tramite un arco che indica la possibile presenza di quella malattia a seguito dell'insorgenza di quel sintomo. Questo schema aiuta a mappare e visualizzare le connessioni tra diverse malattie e i relativi sintomi, fornendo una visione complessa ma chiara delle possibilità diagnostiche.

L'ontologia DO non solo aiuta a stabilire connessioni tra malattie e sintomi, ma consente anche di categorizzare e raggruppare le malattie in base a caratteristiche comuni, come il tipo di patologia e le modalità di presentazione dei sintomi.

L'applicazione non si limita a suggerire una lista di malattie possibili, ma aiuta anche a restringere le opzioni, proponendo diagnosi più mirate in base alla combinazione di sintomi inseriti dall'utente.

## 2 Diagramma delle classi

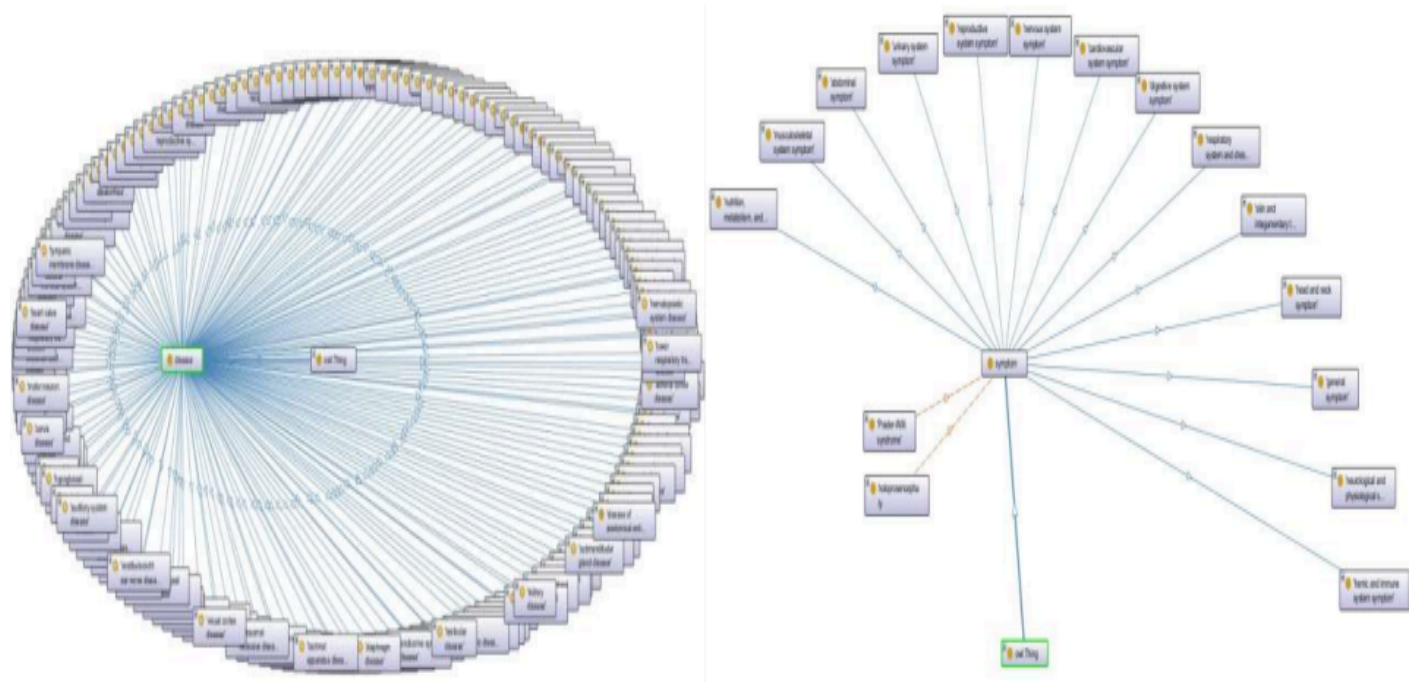
Il sistema di gestione della conoscenza implementato all'interno dell'applicazione è basato su una **Knowledge Base**, la base di conoscenza generata considera due classi principali, ognuna con un ruolo specifico nel rappresentare le entità che sono fondamentali per l'analisi dei dati: la **classe Disease** e la **classe Symptom**.

La **classe Disease** rappresenta le varie malattie o condizioni patologiche che possono essere diagnosticate, ogni istanza di questa classe corrisponde a una malattia specifica, e all'interno della classe, vengono memorizzati dettagli cruciali riguardanti ogni malattia, come i sintomi associati, la durata del decorso, e altre informazioni pertinenti.

La **classe Symptom**, invece, rappresenta i vari sintomi che un individuo può manifestare, ogni istanza di questa classe descrive un sintomo specifico. I sintomi vengono associati alle malattie attraverso relazioni che descrivono come e quando determinati segni clinici siano indicatori di una patologia.

La connessione tra queste due classi è stabilita tramite una **proprietà** chiamata **has\_symptom**, la quale descrive la relazione tra una malattia e i sintomi che la caratterizzano, grazie ad essa è possibile stabilire un legame diretto tra ogni malattia e i suoi sintomi associati, creando una mappa che collega gli effetti fisici (sintomi) alle cause sottostanti (malattie).

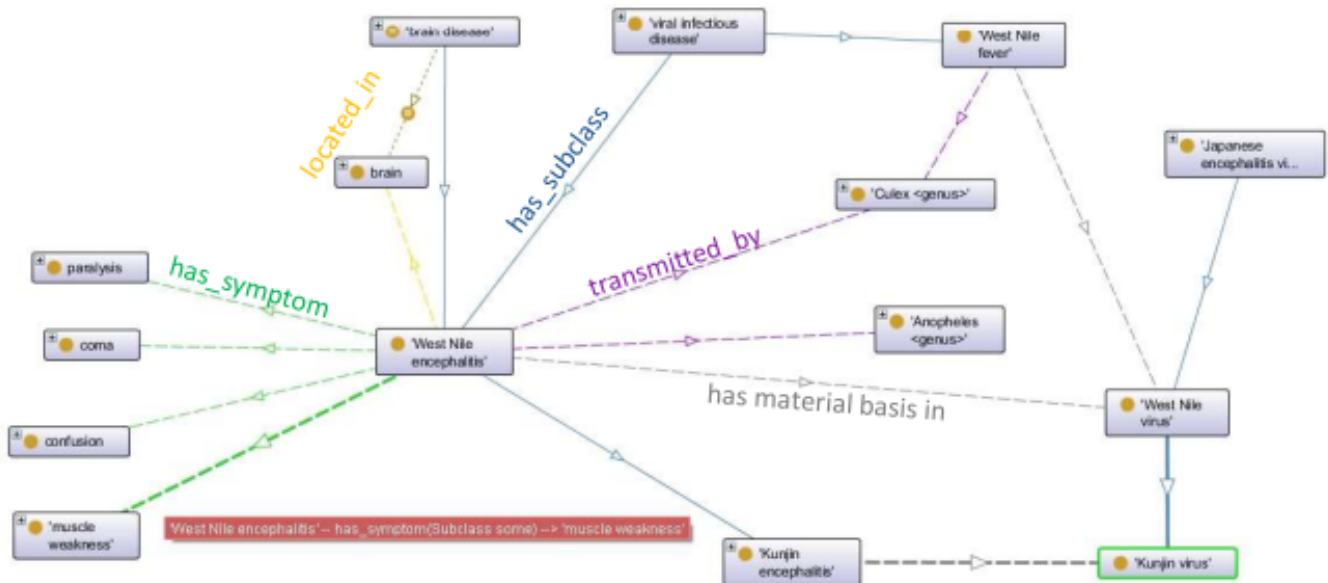
**Rappresentazione grafica:**



Per illustrare meglio come funziona questa relazione, consideriamo un esempio specifico: la malattia «West Nile encephalitis».

La relazione **has\_symptom** (individuata nel diagramma in verde) collega questa malattia a una serie di sintomi.

Questi sintomi sono tipici della West Nile encephalitis, e grazie alla connessione tra la classe **Disease** e la classe **Symptom**, è possibile tracciare un percorso che consenta di associare facilmente i sintomi alla malattia.



Il symptom analyzer a due principali compiti:

1. Visualizzare le malattie più probabili a seconda dei sintomi inseriti dall'utente
2. Dare una spiegazione del risultato

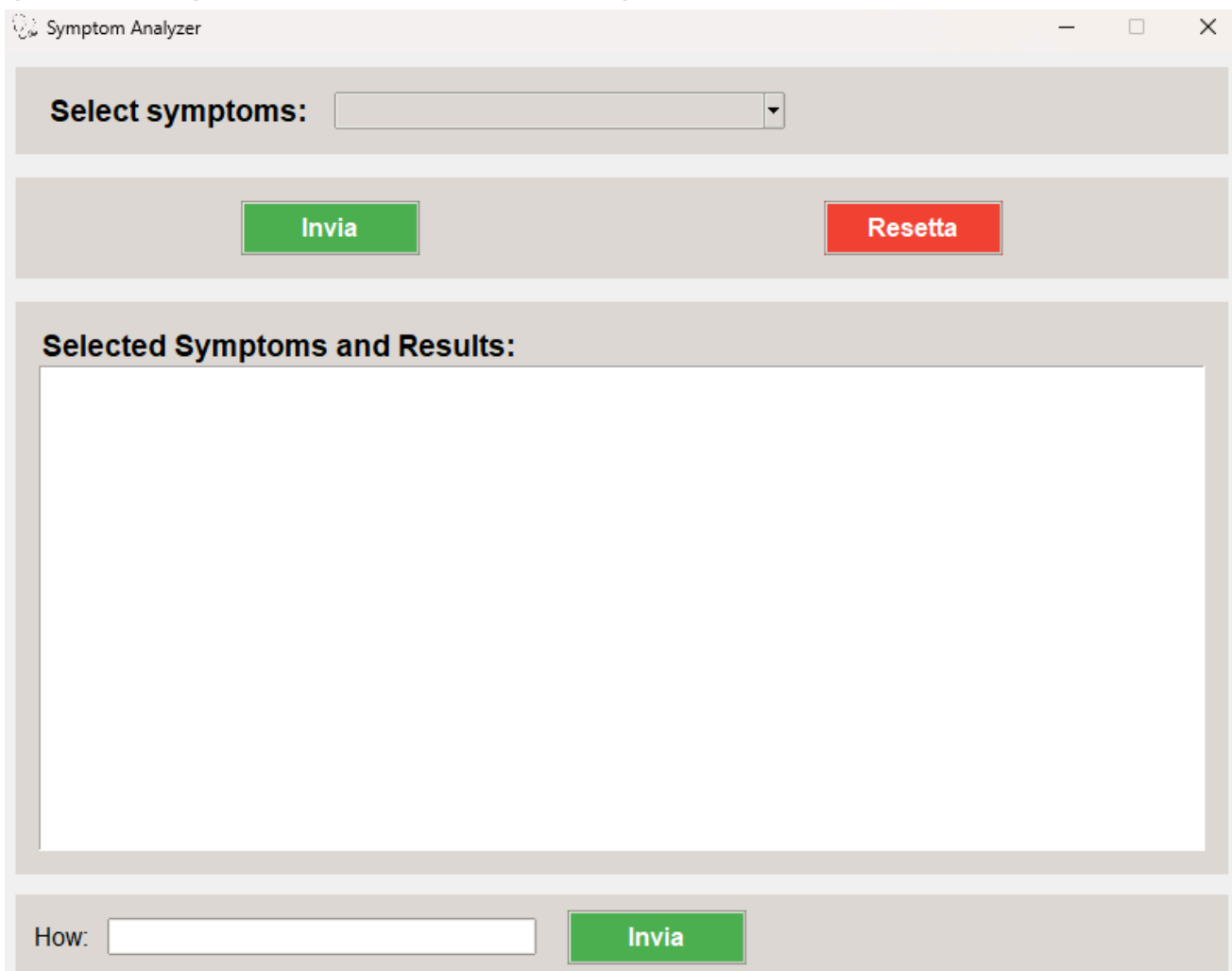
### 3 Struttura dell'interfaccia:

L'interfaccia utente è semplice ed essenziale: Tramite il pulsante «Select symptoms» viene proposta all'utente la lista di tutti i possibili sintomi all'interno della KB, dai quali sceglierà quelli d'interesse.

Quelli scelti, appariranno all'interno della console, ed una volta scelti tutti, l'utente può:

-premere il pulsante «SUBMIT» per far partire l'inferenza e dunque visualizzare il risultato sulla console.

-premere il pulsante «clear console» per resettare la console



The screenshot shows a web application titled "Symptom Analyzer". It features a "Select symptoms:" label followed by a dropdown menu. Below this are two buttons: a green "Invia" button and a red "Resetta" button. A large section titled "Selected Symptoms and Results:" contains a large, empty white box. At the bottom, there is a "How:" label, a text input field, and a green "Invia" button.



## 4 Caso d'uso:

Per avviare il programma, è necessario partire da interfaccia.py, dopodichè come caso d'uso, l'utente inserisce per esempio come sintomi: coma, confusion, muscle weakness e paralysis.

Il sistema restituisce le malattie più probabili in base a quei sintomi.

In questo caso la «West Nile encephalitis» ha la più alta probabilità di essere quella di cui soffre l'utente perché come abbiamo visto prima, questa malattia presenta esattamente i sintomi che l'utente ha inserito.

Infatti nell'ontologia:

The screenshot shows a window titled "Symptom Analyzer". At the top, there is a label "Select symptoms:" followed by a text input field containing "paralysis". Below this, there are two buttons: "Invia" (green) and "Resetta" (red). The main area of the window is titled "Selected Symptoms and Results:" and contains a text area with the text "coma + confusion + muscle weakness + paralysis". Below this, there is a section titled "--- DIAGNOSIS RESULT ---" which contains a table of diseases and their probabilities. The table has two columns: "Disease" and "Probability". The diseases listed are: West Nile encephalitis (26.67%), Japanese encephalitis (13.33%), La Crosse encephalitis (13.33%), St. Louis encephalitis (13.33%), tropical spastic paraparesis (6.67%), staphyloenterotoxemia (6.67%), bagassosis (6.67%), bulbospinal polio (6.67%), and Western equine encephalitis (6.67%). At the bottom of the window, there is a label "How:" followed by a text input field and a green "Invia" button.

Disease	Probability
West Nile encephalitis	26.67%
Japanese encephalitis	13.33%
La Crosse encephalitis	13.33%
St. Louis encephalitis	13.33%
tropical spastic paraparesis	6.67%
staphyloenterotoxemia	6.67%
bagassosis	6.67%
bulbospinal polio	6.67%
Western equine encephalitis	6.67%

## How question

L'inferenza del risultato è prodotta tramite un ragionamento invisibile agli occhi dell'utente, il quale però potrebbe essere interessato a scoprire più informazioni riguardanti la soluzione.

Per questo motivo è stata implementata anche la domanda HOW, che risponde all'esigenza dell'utente di sapere come una clausola è stata restituita. Ecco un'esempio grafico di quello che viene visualizzato:

Symptom Analyzer

Select symptoms:

Invia

Resetta

Selected Symptoms and Results:

coma + confusion + muscle weakness + paralysis

--- DIAGNOSIS RESULT ---

Disease	Probability
West Nile encephalitis	26.67%
Japanese encephalitis	13.33%
La Crosse encephalitis	13.33%
St. Louis encephalitis	13.33%
tropical spastic paraparesis	6.67%
staphyloenterotoxemia	6.67%
bagassosis	6.67%
bulbospinal polio	6.67%
Western equine encephalitis	6.67%

How

Clause used to prove Western equine encephalitis is:  
Western equine encephalitis <- muscle weakness.

OK

How:

Invia

## 5 Architettura

Per gestire la conoscenza come un insieme di clausole definite sono state utilizzate le classi create da David L. Poole e Alan K. Mackworth prese dal sito [aipython.org](http://aipython.org).

Ovvero le classi `logicProblem.py` e `logicBottomUp.py` che definiscono la KB come un insieme di clausole definite, e implementano la procedura di inferenza bottom-up.

Per la lettura dell'ontologia come già detto, è stato utilizzato il modulo python `owlready2`, che fornisce tutte le funzioni necessarie per la manipolazione di ontologie.

A causa di problemi con il parser però l'ontologia è stata modificata tramite l'utilizzo di Protégé (in particolare lo splitter in `owlready2` non riusciva a ritrovare correttamente le triple a causa delle entità e relazioni all'interno dell'ontologia catalogate come "obsolete" da DO stessa).

E' stato dunque possibile esportare l'ontologia rimodellata nel file `do_inferred` il quale viene poi utilizzato all'interno del programma.

La KB quindi viene creata a partire dalla classe `logicProblem.py`, che definisce le clausole definite, un atomo askable, la risposta yes, e la KB stessa.

Tramite questa logica, la KB avrebbe dovuto avere una conoscenza completa, perché una malattia ha un insieme ben definito di sintomi, e solo quei sintomi possono manifestarsi per far sì che quella malattia venga identificata, questa implementazione avrebbe prodotto un sistema eccessivamente selettivo, che avrebbe rilevato malattie solo nel caso in cui l'utente avesse inserito gli esatti sintomi, senza alcuna tolleranza (come per esempio un sintomo falso positivo), per questo motivo, il sistema sfrutta un modello probabilistico, che assegna ad ogni malattia appartenente al punto fisso della KB uno score (per ogni malattia appartenente al punto fisso lo score è tale che sia compreso tra 0 ed 1 e la somma degli score delle malattie appartenenti al punto fisso sia uguale ad 1).

Questo score indica la probabilità che l'utente ha di essere affetto da quella malattia.

La probabilità viene calcolata sul numero dei sintomi in comune tra la malattia presa in considerazione ed i sintomi selezionati dall'utente.

Per esempio se definiamo  $S(u)$  l'insieme dei sintomi dell'utente, e con  $S(m)$  l'insieme dei sintomi che caratterizzano la malattia, allora la probabilità che l'utente abbia

contratto la malattia  $m$  è denotata da: 
$$\frac{|S(m)|}{|S(u)|}$$

Dunque se sono stati inseriti tutti e soli i sintomi della malattia  $m$  (nonostante questo non escluda il fatto che possano essere estratte anche altre malattie) la probabilità di aver contratto la malattia  $m$  sarà sicuramente maggiore o al più uguale alla probabilità di aver contratto un'altra malattia del punto fisso.

## 6 Conclusioni

In conclusione, il progetto è uno strumento per poter fare un'analisi delle malattie associate partendo dai sintomi in maniera probabilistica, creando così uno strumento per gli utenti che vogliono informarsi su tale malattie per una pre autodiagnosi anche se probabilistica (chiaramente da non sostituire con un dottore).

### Riferimenti Bibliografici:

- I. D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents, Cambridge University Press. 3rd ed.
- II. dispense e altro materiale;