

WordNet en Prolog

Alejandro Camacho Pérez
Grupo C-312

Índice

1. Resumen	2
2. Descripción	3
2.1. Interfaz gráfica	3
2.1.1. Entrada de datos:	4
2.1.2. Botón de consulta:	5
2.1.3. Resultados:	6
2.1.4. Operadores	7
2.2. Consultas	7
2.2.1. Assertions	7
2.2.2. Entailment	7
2.2.3. Meronym/Holonym	8
2.2.4. Attribute	8
2.2.5. Aditonal information	8
2.2.6. Pertains	8
2.2.7. Hypernym	8
2.2.8. Similarity	9
2.2.9. Caused	9
2.2.10. Antonym	9
2.2.11. Participle	9
2.3. Implementación	9
2.3.1. gui.py	10
2.3.2. prolog_api.py	11
2.3.3. main.py	13
3. Bibliografía	13

1. Resumen

En este trabajo se ha implementado una interfaz en Python para acceder a la base de datos de WordNet en Prolog, de forma que se puedan realizar consultas de forma sencilla y obtener los resultados de forma legible. Para ello se ha utilizado la librería `pyswip` que permite la comunicación entre Python y Prolog, así como la librería `tkinter` para la interfaz gráfica.

La interfaz permite realizar 11 tipos de consultas diferentes, que se describen en la sección 2.2. Para cada una de ellas se ha implementado una función en Python que se encarga de realizar la consulta en Prolog y devolver los resultados.

2. Descripción

2.1. Interfaz gráfica

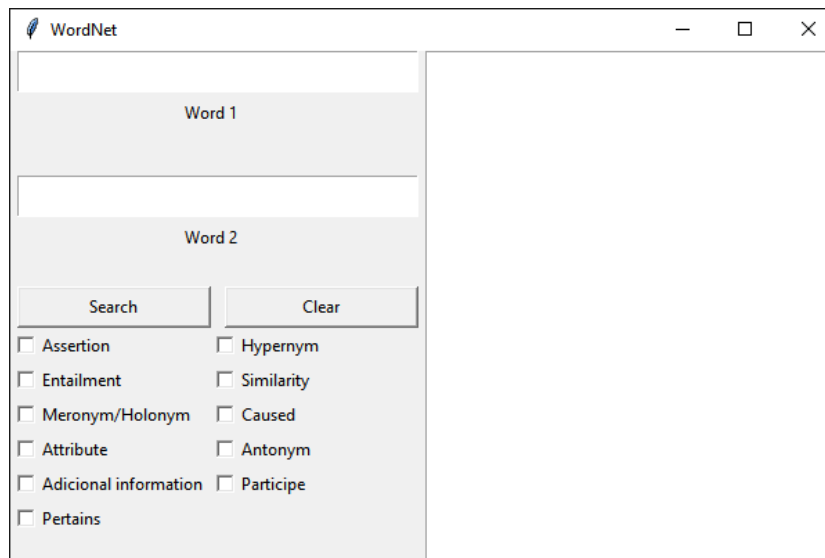
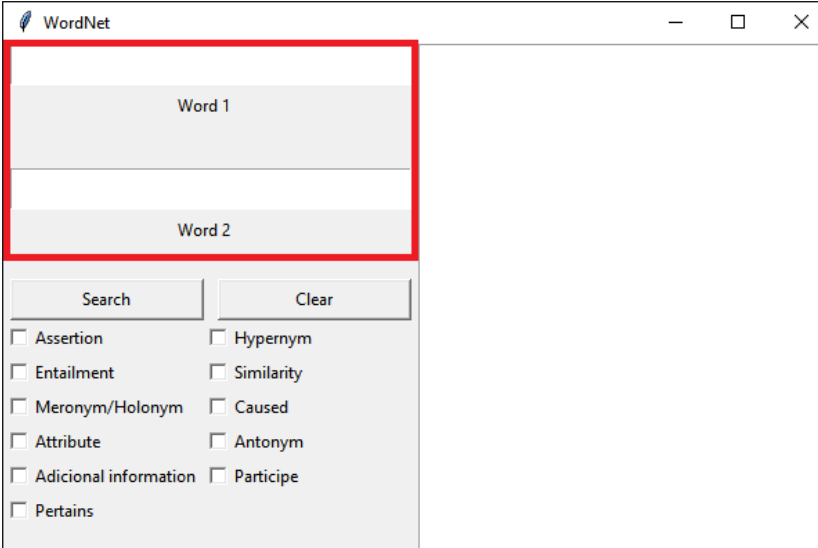


Figura 1: Interfaz gráfica

La interfaz gráfica se ha implementado utilizando la librería `tkinter` de Python. Se ha creado una clase GUI que contiene todos los elementos de la interfaz, y que se inicializa en el fichero `main.py`. La interfaz se divide en tres partes:

2.1.1. Entrada de datos:



The screenshot shows a window titled "WordNet" with a standard Windows-style title bar (minimize, maximize, close buttons). The main content area is divided into two input fields, "Word 1" and "Word 2", which are highlighted by a red rectangular border. Below these fields are two buttons: "Search" and "Clear". Underneath the buttons is a list of relationship types, each preceded by an unchecked checkbox:

- ☐ Assertion
- ☐ Entailment
- ☐ Meronym/Holonym
- ☐ Attribute
- ☐ Adicional information
- ☐ Pertains
- ☐ Hypernym
- ☐ Similarity
- ☐ Caused
- ☐ Antonym
- ☐ Participle

Figura 2: Entrada de datos

En esta parte se introducen los datos necesarios para realizar la consulta. El tipo de consulta determina los datos que se deben introducir. Por ejemplo, para la consulta **hiperónimos** se debe introducir un sustantivo, mientras que para la consulta **antónimos** se debe introducir un adjetivo. El programa no permite realizar consultas con datos incorrectos, es decir, si se introduce un adjetivo en una consulta que requiere un sustantivo, se mostrará un mensaje de error.

2.1.2. Botón de consulta:

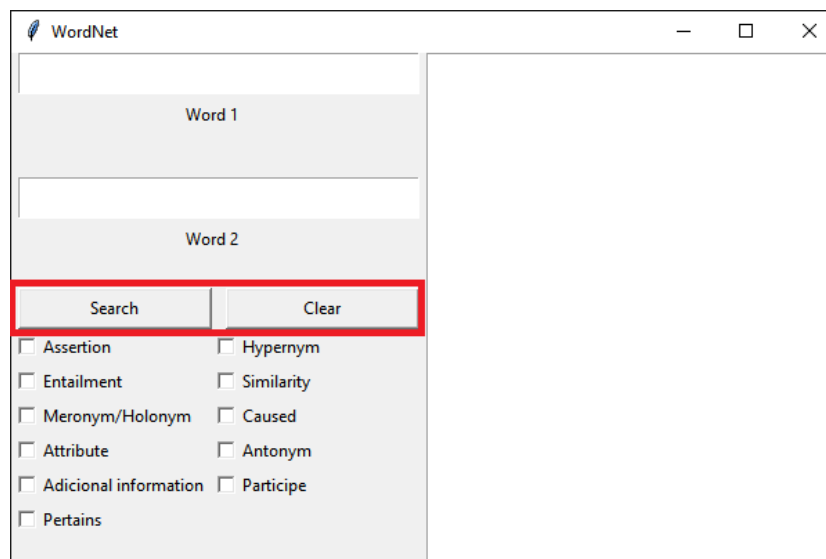


Figura 3: Botón de consulta

El botón **Search** se utiliza para realizar la consulta con los datos introducidos y el operador seleccionado.

El botón **Clear** se utiliza para limpiar los datos introducidos y los resultados de la consulta.

2.1.3. Resultados:

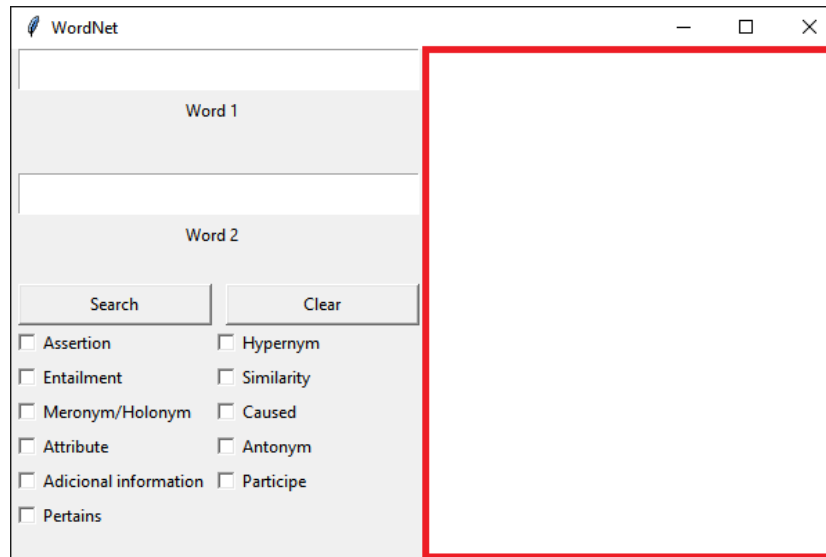


Figura 4: Resultados

En esta región se muestran los resultados de la consulta, así como los mensajes de error en caso de que se produzcan.

2.1.4. Operadores

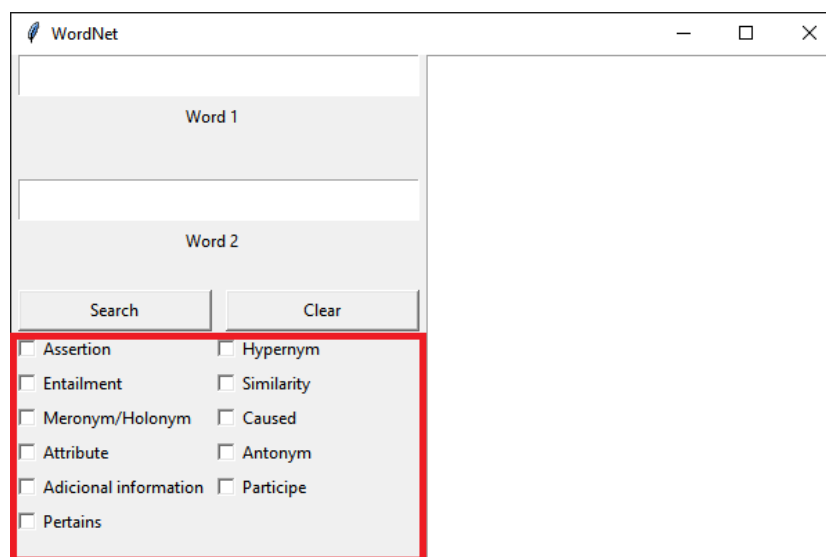


Figura 5: Operadores

Para poder realizar una consulta, es necesario seleccionar el operador a utilizar, que determina el tipo de consulta que se va a realizar.

2.2. Consultas

Se han implementado 11 tipos de consultas diferentes, que se describen a continuación. Para cada una de ellas se ha implementado una función en Python que se encarga de realizar la consulta en Prolog y devolver los resultados.

2.2.1. Assertions

Se utiliza para encontrar todos los significados posibles de la palabra especificada en la primera entrada.

2.2.2. Entailment

Se utiliza para encontrar todos los verbos que se vinculan con el verbo especificado en la primera entrada. Si sólo se especifica la segunda entrada,

se utiliza para encontrar todos los verbos de los cuales se deriva de su acción el verbo especificado. En caso de que sean utilizadas ambas entradas, se comprueba si los verbos dados están vinculados.

2.2.3. Meronym/Holonym

Se utiliza para encontrar los merónimos o holónimos de la palabra especificada en la primera o la segunda entrada respectivamente. En caso de que sean utilizadas ambas entradas, se comprueba si la palabra especificada en la primera entrada es un merónimo de la palabra especificada en la segunda entrada.

2.2.4. Attribute

Se utiliza para encontrar los adjetivos que hacen de atributo al sustantivo especificado en la primera entrada. En caso de que se especifique la segunda entrada, se buscan los sustantivos de los cuales ese adjetivo hace de atributo. Si ambas entradas son especificadas, se comprueba si el adjetivo especificado en la primera entrada hace de atributo al sustantivo especificado en la segunda entrada.

2.2.5. Additional information

Se utiliza para encontrar las palabras que aportan información adicional al verbo o adjetivo especificado en la primera entrada.

2.2.6. Pertains

Si en la primera entrada se especifica un adjetivo, el resultado será un sustantivo o adjetivo que se relaciona con el adjetivo especificado. Si en la primera entrada se especifica un adverbio, el resultado será un adjetivo del cual se deriva el adverbio especificado.

2.2.7. Hypernym

Se utiliza para encontrar los hiperónimos del sustantivo especificado en la primera entrada. En caso de que se especifique la segunda entrada, se buscan los sustantivos de los cuales el sustantivo especificado en la primera entrada es hiperónimo. Si ambas entradas son especificadas, se comprueba si

el sustantivo especificado en la primera entrada es hiperónimo del sustantivo especificado en la segunda entrada.

2.2.8. Similarity

Se utiliza para encontrar los adjetivos con significado similar al adjetivo especificado en una de las entradas. Si se especifican ambas entradas, se comprueba si los adjetivos especificados son similares.

2.2.9. Caused

Se utiliza para encontrar los verbos que causados por el verbo especificado en la primera entrada. Si se especifica la segunda entrada, se buscan los verbos que causan el verbo especificado en la primera entrada. Si se especifican ambas entradas, se comprueba si el verbo especificado en la primera entrada causa el verbo especificado en la segunda entrada.

2.2.10. Antonym

Se utiliza para encontrar los antónimos del adjetivo especificado en la primera entrada. Si se especifica la segunda entrada, se buscan los adjetivos que son antónimos del adjetivo especificado en la primera entrada. Si se especifican ambas entradas, se comprueba si el adjetivo especificado en la primera entrada es antónimo del adjetivo especificado en la segunda entrada.

2.2.11. Participle

Se utiliza para encontrar los participios del verbo especificado en la primera entrada. Si se especifica la segunda entrada, se buscan los verbos de los cuales el verbo especificado en la primera entrada es participio. Si se especifican ambas entradas, se comprueba si el verbo especificado en la primera entrada es participio del verbo especificado en la segunda entrada.

2.3. Implementación

El proyecto se divide en 3 archivos principales: `main.py`, `gui.py` y `prolog_api.py`. En el archivo `main.py` se inicializa la interfaz gráfica y se definen las funciones que se ejecutan al pulsar el botón de consulta. En el archivo `gui.py` se define la clase GUI que contiene todos los elementos de la interfaz gráfica. En

el archivo `prolog.py` se define la clase `Consulter` que contiene las funciones que se encargan de realizar las consultas en Prolog.

2.3.1. `gui.py`

Este código en Python es una implementación de una interfaz gráfica de usuario (GUI) utilizando la biblioteca `tkinter` para consultas en un programa basado en Prolog (se hace referencia a `prolog_api`). La interfaz consta de botones de selección (`CheckButton`) que representan diferentes opciones y dos campos de entrada de texto para ingresar palabras. Además, se incluyen botones de búsqueda (`Search`) y limpieza (`Clear`), junto con un área de texto para mostrar los resultados.

1. Se importan las bibliotecas necesarias, como `Enum` de `enum`, `tkinter` y algunos elementos personalizados (`Consulter` y `Operator`).
2. Se define la clase `GUI` que representa la interfaz de usuario. El constructor `__init__` inicializa varios atributos y crea la ventana principal de la interfaz.
3. El método `run` inicia el bucle principal de la interfaz gráfica (`mainloop`).
4. El método `place` organiza los elementos gráficos en la ventana según cuadrantes y tamaños relativos.
5. Se definen métodos para el manejo de eventos, como el redimensionamiento de la ventana (`resize_result_text`), la creación de botones de verificación (`create_check_buttons`), y la manipulación del estado de los botones de verificación (`check_box_behavior`).
6. El método `select_text` actualiza la información asociada a las entradas de texto según el estado de los botones de verificación.
7. El método `show_result` actualiza el área de texto con el resultado de una consulta.
8. Los métodos `search_behavior` y `clear_behavior` están asociados a los botones de búsqueda y limpieza, respectivamente. El primero realiza una consulta al sistema basado en Prolog y actualiza la interfaz con el resultado. El segundo restablece el estado de la interfaz a valores iniciales.

2.3.2. prolog_api.py

En el archivo `prolog_api.py` se definen las clases `Consulter`, `Operator` y `WordInfo`.

La clase `Consulter` se encarga de realizar las consultas en Prolog. Para ello utiliza la librería `pyswip` que permite la comunicación entre Python y Prolog. En el constructor de la clase, se inicializa el objeto `Prolog` que se encarga de realizar las consultas, y se cargan en memoria los archivos de la base de datos de WordNet con el método `load_consults`. Para realizar una consulta a la base de datos, se cuenta con el método `receive_query` que recibe como parámetro el operador a utilizar y las palabras de la consulta. Esta query es almacenada en la clase para su posterior uso por el método `process_query` que se encarga de obtener los resultados de la consulta.

Al realizar el llamado para procesar la query, el método primero busca la información necesaria en la base de datos de ambas palabras introducidas, para luego seleccionar según el operador, que método se utilizará para procesar la query. Cada método devuelve un string con los resultados de la consulta, que es luego almacenado en la variable `result_string` de la clase, de forma tal que siempre se puede acceder al último resultado procesado. Para corregir errores de paralelismo de la librería `pyswip`, se utiliza el método `make_consult`, quien recibe el string a consultar a los archivos prolog, y espera a que el resultado sea procesado para devolverlo en una lista.

Los métodos específicos para procesar cada tipo de consulta, siguen el siguiente patrón:

1. Comprueban si las palabras introducidas son válidas, es decir, si se encuentran en la base de datos. En caso de que no lo sean, se devuelve un mensaje de error.
2. Se crea la query a consultar en Prolog, utilizando los datos obtenidos de la base de datos.
3. Se realiza la consulta en Prolog utilizando el método `make_consult`.
4. Se procesa la lista devuelta para dar un resultado legible y claro.

5. Se devuelve el resultado.

Los métodos utilizados son:

- `assertion()`
- `similarity_1_to_2()`
- `similarity_1_to_all(word_selector)`
- `antonym_1_to_2()`
- `antonym_1_to_all(word_selector)`
- `is_hyponym()`
- `hyponym_of()`
- `inverse_hyponym()`
- `is_entailment()`
- `entailment_of()`
- `inverse_entailment()`
- `is_meronym_holonym()`
- `mer_hol(function,operator)`
- `caused()`
- `is_attribute()`
- `attribute_of(word_selector)`
- `sa()`
- `is_participle()`
- `participle_of(word_selector)`
- `pertains()`

Cada uno de ellos está bien documentado en el código.

El enum `Operator` se utiliza para almacenar los operadores disponibles.

La clase `WordInfo` es una representación de la información asociada a una palabra en WordNet. Esta clase tiene diversos atributos que almacenan detalles como los identificadores de synsets (`synset_id_list`), tipos de palabra (`word_type_list`), números asociados por cada synset (`word_number_list`), grado de sensibilidad de la palabra (`word_sense_list`), significado de la palabra en un synset (`gloss_list`), cantidad de veces que aparece la palabra (`tag_count_list`), y un indicador booleano (`exist`) que señala si la palabra existe o no en la base de datos.

El constructor `__init__(self, word: str)` inicializa la instancia de la clase, tomando la palabra como parámetro y asignándola al atributo `word`. Los demás atributos se inicializan como listas vacías y el indicador de existencia se establece en `False`.

El método `__str__(self)` proporciona una representación de cadena de la palabra, devolviendo simplemente el valor del atributo `word`.

2.3.3. `main.py`

El archivo `main.py` es donde se crea la instancia de la clase `GUI` y se ejecuta el método `run` para iniciar la interfaz gráfica.

3. Bibliografía

- Sarah Witzig 'Accessing WordNet from Prolog' (2003)
- Documentación de WordNet