Conocimiento y Razonamiento Automatizado

Analizador sintáctico de oraciones

Grado en Ingeniería Informática
Universidad de Alcalá

Pablo García García
Álvaro Jesús Martínez Parra
Alejandro Raboso Vindel

27 de abril de 2023

Índice general

Capítulo 1

Introducción y objetivos

1.1. Análisis de oraciones

A modo de introducción, el objetivo principal de la práctica es conseguir analizar sintácticamente oraciones que contengan ciertas palabras definidas, y que estén formadas por una serie de estructuras sintácticas comunes. Lo que se quiere conseguir es pasar una oración y devolver en forma de árbol de constituyentes la estructura sintáctica de la misma. Un ejemplo sería, dada la oración: *"El pitufo que es azul, toca el piano"*, queremos obtener el árbol que observamos en la Figura 1.1.

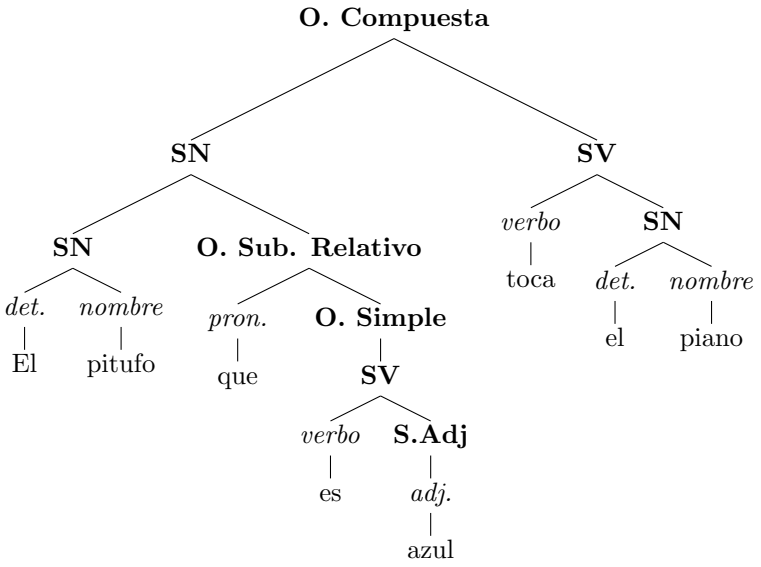


Figura 1.1: Primer ejemplo

1.2. Simplificación de oraciones

Como segundo objetivo principal, lo que queremos es al encontrarnos con oraciones compuestas, es descomponerla en varias oraciones simples. Un ejemplo sería dada la oración: *"El*

pitufo que es azul, toca el piano", queremos ser capaces de descomponerla en:

- *"El pitufo es azul"*
- *"El pitufo toca el piano"*

Además, tendremos como objetivos extra una serie de mejoras que hemos propuesto y que se explicarán en el Capítulo 4.

Capítulo 2

Análisis sintáctico de oraciones

Este analizador funcionará para oraciones que sigan una estructura similar a las propuestas por el enunciado, en general serán:

- **Oraciones simples**: estas son oraciones que contienen un solo verbo. Podemos distinguir entre oraciones simples con sujeto y predicado, o solo con predicado.
- **Oraciones compuestas**: son oraciones que tienen más de un verbo, salvo que el verbo esté en infinitivo y haga la función de un nombre. En caso de que tengan más de un verbo, seguirán la estructura de estos dos tipos de oraciones.
  - **Oraciones coordinadas**: son dos oraciones separadas por una conjunción.
  - **Oraciones subordinadas de relativo**: siempre va precedida por un pronombre de relativo, en nuestro caso *que*. A este pronombre le sigue otra oración.

A su vez, las oraciones deberán estar compuestas por los siguientes grupos sintácticos o sintagmas:

- Sintagma nominal
- Sintagma adjetival
- Sintagma adverbial
- Sintagma preposicional
- Sintagma verbal

Finalmente, estos grupos sintácticos deberán estar compuestos de los siguientes tipos de términos:

- Determinantes
- Nombres
- Nombres propios

- Verbos
- Adjetivos
- Adverbios
- Conjunciones
- Preposiciones
- Pronombre (de relativo)

2.1. Diccionario

Para poder realizar el análisis sintáctico de las oraciones es necesaria la creación de un diccionario con todas las palabras de la lengua que se vayan a usar. Se podrán introducir oraciones que sigan las estructuras previamente comentadas y que contengan únicamente palabras que estén en este diccionario. Si se quiere trabajar con palabras nuevas, deberán ser añadidas a este, indicando el tipo de término que es.

Este diccionario lo crearemos con las palabras separadas por su tipo, si son nombres propios, sustantivos, adjetivos, etc. Es necesario guardar las palabras con su tipo para poder realizar un análisis correcto, debido a que será necesario saber en qué orden están en las oraciones, para así conocer la estructura que tendrá la oración. Por ejemplo, para representar el término *mesa* que es un nombre, escribiremos `n('mesa')`. Las abreviaturas para cada tipo de término es:

- `np` → nombre propio
- `n` → nombre
- `v` → verbo
- `inf` → infinitivo
- `adj` → adjetivo
- `det` → determinante
- `conj` → conjunción
- `prep` → preposición
- `adv` → adverbio
- `pro` → pronombre

2.2. Gramática

Una vez definidas todas las palabras de las que se dispone, así como su tipo de término correspondiente, deberá definirse la estructura de los sintagmas que están formados por estos términos. Todo lo mencionado en este apartado queda recogido en el archivo gramatica.pl.

Lo que se ha realizado para cada tipo de estructura sintáctica, ha sido escribir los términos o sintagmas de los que pueda estar compuesto, en su correspondiente orden. Un ejemplo es el siguiente, donde se define cómo debe ser un sintagma nominal:

```
20 %---SINTAGMAS---
21 % Sintagma nominal
22 grupo_nominal(gn(N)) --> nombre_propio(N).
23 grupo_nominal(gn(NA, C, NB)) --> nombre_propio(NA), conjuncion(C),
       nombre_propio(NB).
24 grupo_nominal(gn(N)) --> nombre(N).
25 grupo_nominal(gn(D, N)) --> determinante(D), nombre(N).
26 grupo_nominal(gn(N, GA)) --> nombre(N), grupo_adjetival(GA).
27 grupo_nominal(gn(D, N, GA)) --> determinante(D), nombre(N),
       grupo_adjetival(GA).
28 grupo_nominal(gn(D, N, GP)) --> determinante(D), nombre(N),
       grupo_preposicional(GP).
29 grupo_nominal(gn(I, GN)) --> infinitivo(I), grupo_nominal(GN).
```

Código 2.1: Definición de sintagma nominal

De la misma manera se han implementado el resto de sintagmas (tendiendo en cuenta los respectivos órdenes de sus componentes) y la estructura de las oraciones en sí. Sin embargo, ha sido necesario definir otros sintagmas auxiliares (que lingüísticamente no existen) que nos ayudarán a llevar a cabo nuestro análisis. Vamos a explicarlos a continuación:

- `grupo_nominal_compuesto`: está formado por un sintagma nominal y una oración subordinada de relativo. Se ha utilizado para detectar que se está ante una oración compuesta, en este caso utilizada como complemento de un nombre.
- `grupo_verbal_compuesto`: sigue la misma idea que `grupo_nominal_compuesto`, pero en este caso la oración subordinada es un complemento del propio sintagma verbal.
- `infinitivo`: este no es un grupo, si no un término. La función que desempeña es representan verbos como nombre, cuando estos se encuentran en su forma de infinitivo. Por ejemplo, en la oración *"Sirve para escribir documentos"*, en la parte señalada, *escribir* hace la función de nombre pese a ser un verbo. En este caso, no se consideraría oración compuesta al comportarse este como un nombre, y haber un único nombre en la oración.

Además, se ha creado algún predicado auxiliar, como `quitarComas/2`, que nos ayuda a eliminar comas de los terminales que las contengan, para poder empezar a realizar el análisis con palabras de nuestro diccionario.

Finalmente, nos encontramos con el predicado `analizar/2`, que recibe una lista de terminales en la variable `L`, y devuelve en `X` un `compound` con la definición resultante del análisis sintáctico, listo para ser tratado. El predicado `analizar/2` quita las comas de la oración inicial, la analiza, y muestra con ASCII Art el árbol resultante con ayuda del programa proporcionado `draw.pl`. Una vez se obtiene el árbol del análisis sintáctico, basta con llamar al predicado `draw/1`, pasando este como parámetro. También ejecutará una serie de mejoras explicadas más adelante.

Para llamar al predicado analizar con, por ejemplo, la oración *"José come patatas fritas."*, deberemos escribir lo siguiente:

```
?- analizar(['JOSÉ', 'come', 'patatas', 'fritas'], X).
```

Capítulo 3

Simplificación de oraciones

Para poder simplificar oraciones compuestas en múltiples oraciones simples, hemos construido los predicados `descomponer/1`, `cogerSujeto/2`, `explorar/2`, ubicados en el archivo `descomponer.pl`.

La idea de esta simplificación se basa en el recorrido recursivo del árbol de análisis sintáctico. Se tendrá que ir recorriendo hasta llegar a los nodos hoja, sobre los que habrá que decidir si hay que imprimirlos y en qué orden, dependiendo de la estructura de la oración. Se sabe que las estructuras sintácticas tienen a lo sumo tres nodos hijos, por lo que habrá que hacer predicados generales para explorar nodos con 0, 1, 2, o 3 hijos. En caso de tener cero hijos, será un nodo hoja a imprimir.

No solo debe tenerse esto en cuenta, pues también se pueden encontrar estructuras que habrá que saber cómo recorrer, con el fin de separar las oraciones simples del resto de estructuras.

Al estar trabajando con `compounds` hacemos uso de los predicados básicos de Prolog que permiten trabajar con esta estructura de datos, `functor/3` y `arg/3`. El primero de ellos, recibe un `compound`, y devuelve en la segunda variable el contenido del nodo raíz de ese `compound`. En la tercera variable devuelve el número de hijos del nodo raíz. El segundo de ellos, se indica en la primera variable el número del hijo[1] que se quiere extraer del `compound` (indicado en la segunda variable), y lo devuelve en la tercera variable.

3.1. `cogerSujeto/2`

En cada oración siempre existe un sujeto general al que se refiere la misma. En caso de existir, este se encontrará en el primer sintagma nominal de la oración. El objetivo de este predicado es extraer ese sintagma nominal (sujeto general), para, en caso de obtener una oración simple sin sujeto, saber a quién hace referencia.

[1] `arg/3` empieza a contar desde 1.

3.2. explorar/2

Este predicado va a recorrer el árbol del análisis sintáctico, tal y como se ha explicado previamente. Existen algunos casos especiales donde el recorrido del árbol tenga que ser ligeramente diferente debido a la estructura del subárbol que se esté recorriendo; y así obtener una descomposición coherente de la oración compuesta general. Estas estructuras son:

- **Oración subordinada de relativo**
  - En caso de tener un predicado donde su sintagma nominal tenga una oración subordinada de relativo, habrá que imprimir el verbo seguido del subgrupo nominal (sin contar la oración subordinada) y así se obtiene una oración simple. La segunda oración, saldrá de imprimir la oración subordinada sin el pronombre y con el sujeto general de la oración en caso de que esté omitido, en caso contrario se imprime la oración.
  - En caso de tener un sujeto donde haya una oración subordinada como complemento, habrá que imprimir el sujeto al que se hace referencia con la oración subordinada sin el pronombre, y después el predicado con el sujeto al que hace referencia la oración (sin la oración subordinada).
- **Oración coordinada**
  - En caso de tener una oración coordinada sin un sujeto al comienzo de la misma, se deberá imprimir cada oración simple que forma la coordinada con el sujeto general de la oración.
  - En caso contrario, se deberá imprimir cada oración simple que forma la coordinada con el sujeto de la oración coordinada.
- **Oración simple con sujeto omitido**
  - En caso de no haber un sujeto explícito en la oración, se deberá imprimir el sujeto general de la oración seguido del predicado de la oración.

En caso de que no se encuentre ninguno de estos casos, se seguirá la exploración del árbol de izquierda a derecha, tal y como se especificó en un inicio.

3.3. descomponer/1

Este es el predicado que recibe el análisis sintáctico de la oración como `compound`, y hace uso de los dos predicados explicados previamente para descomponer la oración en varias simples. Una vez encuentra una respuesta correcta, para la búsqueda con el operador corte (`!`).

Capítulo 4

Mejoras

4.1. Traducción a inglés

La primera mejora ha sido implementar un traductor español-inglés, de tal forma que al introducir la oración no solo se analice y descomponga, sino que además se traduzca a este idioma. Para ello nos hemos fijado en la estructura del archivo `traductor.pl` y lo hemos extendido a nuestro diccionario, implementando así esta funcionalidad.

Lo primero de todo ha sido añadir el diccionario de inglés (`diccionario_eng.pl`). En él podemos ver todas las palabras que habíamos definido en el diccionario de español, solo que ahora especificamos la palabra en español y en inglés. De esta forma establecemos la equivalencia en ambos idiomas. Por ejemplo, en el diccionario español tenemos `n(mesa).`, y en el diccionario de inglés tendremos `n(mesa, table).`. Este proceso lo hemos repetido con todas las palabras, sin alterar en ningún momento el tipo de término de la palabra.

Una vez tenemos nuestro diccionario definido, podemos elaborar el traductor en sí. Para ello tendremos que copiar nuestra gramática en español, poniendo delante de todos los predicados un parámetro `esp` que indica que esa gramática está en español. Por ejemplo, `grupo_nominal(gn(det(X), n(Y))):- determinante(X), nombre (Y).` pasaría a ser `grupo_nominal(esp, gn(det(X), n(Y))): -determinante(esp, X), nombre (esp, Y).`. Tendremos que volver a copiar la gramática, pero esta segunda copia con `eng` en vez de `esp`. De esta forma diferenciamos la gramática en inglés.

El objetivo de definir ambas gramáticas es que podemos reestructurar la oración para que esta concuerde con la gramática inglesa. Por ejemplo, en español tenemos determinante, nombre y adjetivo como sintagma nominal. En la gramática inglesa, esta estructura es errónea, siendo esta determinante, adjetivo, nombre. Bastará con cambiar el orden de los predicados dentro de la regla del sintagma nominal en la gramática inglesa. Así, hemos puesto en todos los predicados que sea necesario que el adjetivo vaya antes que el nombre.

Con ambas gramáticas definidas, basta con realizar el análisis sintáctico de la oración en español y luego llamar al predicado de la oración en inglés con el parámetro `eng` seguido del

análisis sintáctico de la oración en español la variable donde se va a almacenar la oración en inglés y una lista vacía que le permitirá recuperarse de errores. El resultado se nos da en una lista con todas las palabras que componen la oración en inglés.

Para no devolver una lista e imprimir por pantalla el resultado, hemos creado un predicado `escribir/1` que lo que hace es ir escribiendo cada elemento de esa lista. Finalmente tenemos el predicado `traducir/2` al que habrá que llamar para realizar toda esta funcionalidad una vez tengamos el análisis sintáctico de la oración que se pretende traducir en español. Bastará con pasar `eng` como primer parámetro (indicando que el idioma al que se quiere traducir es inglés) y el `compound` con el análisis sintáctico en español en el segundo parámetro.

4.2. Conexión entre Prolog y Python

Como mejora, se ha creado un script en Python que nos permita agilizar la introducción de frases a Prolog. Se invocará de la siguiente manera desde la terminal donde se tengan todos los archivos que componen a la práctica:

```
> python[1] analizador.py -f "frase"
```

En `frase` escribimos la frase a analizar, sin punto final, y usando las palabras del diccionario. En el parámetro `f` deberá escribirse `pdf` o `img` en función de si queremos obtener un dibujo del árbol en formato PDF o JPG.

El funcionamiento es muy simple. Lo que hace es recoger la frase introducida por el usuario, y le da el formato de una consulta como la explicada en el final de la Sección 2.2. Una vez se tiene en un string la consulta lista, con ayuda de la librería `pyswipl`, Python puede hacerle consultas a Prolog. Para ello le indicamos que debe hacer `consult` del archivo `gramatica.pl`, y mediante el método `query`, le hacemos la consulta que previamente habíamos preparado, y para la respuesta obtenida invocamos al conversor explicado en la Sección 4.3. Finalmente, volverá a hacer una consulta de descomponer la respuesta previamente obtenida, pero pasando todo a minúsculas, para evitar errores con `functor/3`, pues si hay mayúsculas, interpreta que son variables. Podemos ver un ejemplo en la Figura 4.1.

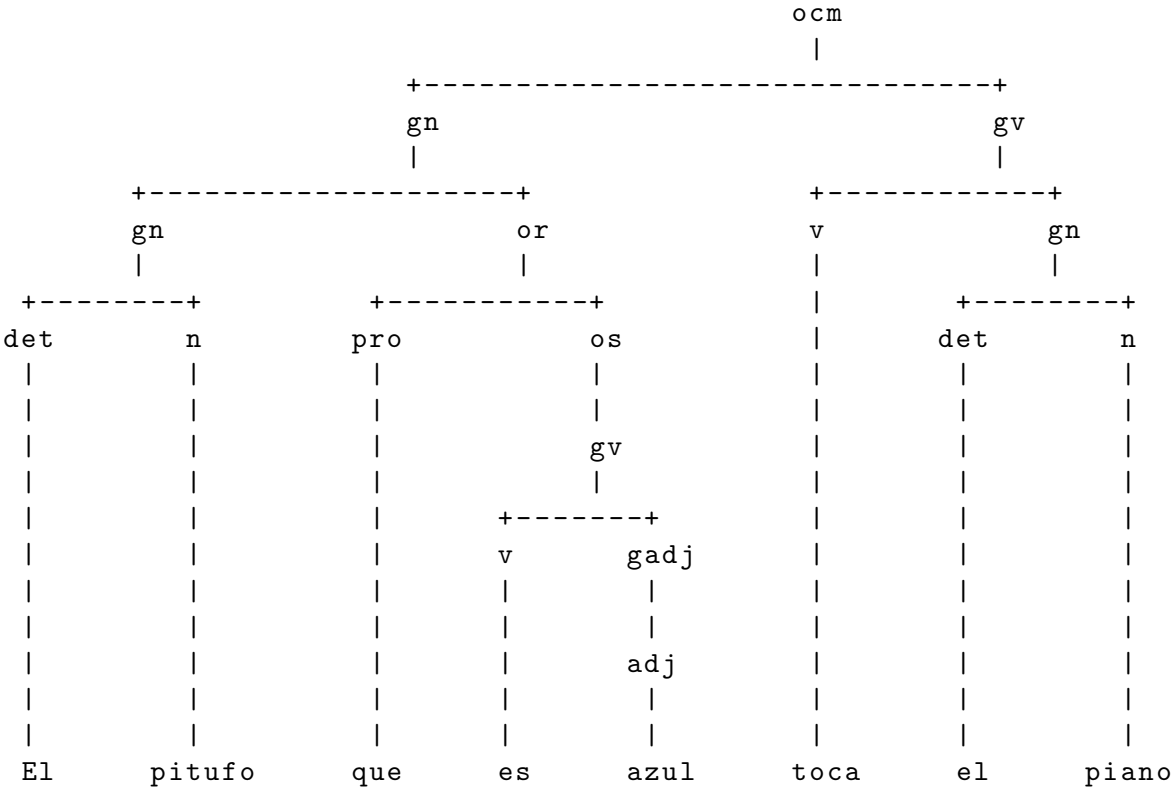
4.3. Árbol sintáctico en LaTeX

Como mejora, se ha optado por generar una mejor visualización del árbol de sintaxis que la ofrecida por el archivo `draw.pl`. Para ello, hemos elegido LaTeX como lenguaje que nos permitiera realizar el dibujo del árbol en tiempo de ejecución dada la sintaxis de Prolog. Las salidas de esta mejora son los diferentes árboles que se han ido mostrando a lo largo de este documento.

Para llevar esto a cabo, nos hemos aprovechado de lo explicado previamente en la Sección 4.2, añadiendo un archivo llamado `Prolog2LaTeX.py` que contiene diversas funciones, siendo

[1]Es posible que puedas tener tu varriable de entorno de Python definida como `py` o `python3`.

```
> python analizador.py -pdf "El pitufo que es azul toca el piano"
```




```
-> Traduccion al ingles
The smurf that is blue plays the piano
-> Descomposicion de la oracion compuesta
el pitufo es azul
el pitufo toca el piano
```

Figura 4.1: Ejemplo de `analizador.py`

`main` la única que necesitamos llamar para generar nuestro dibujo. Como en el programa `analizador.py` podíamos recoger los resultados de nuestras consultas en Prolog en forma de diccionario, pasaremos esta respuesta a la función `main` previamente nombrada y esta hará todo el trabajo.

Expliquemos ahora el funcionamiento general de `Prolog2LaTeX.py`. En este fichero tenemos tres funciones importantes, que se ejecutan en el siguiente orden en la función `main`: `parser`, `compile`, y `show`.

- `parser`: esta función recibe la respuesta de Prolog en el formato de `compound`, y la convierte a la sintaxis que utiliza un objeto `\Tree` del paquete `qtree` de LaTeX. Como la sintaxis es la siguiente `[.P1 [.HIJO1_P1 [.HIJO1_HIJO1_P1] [.HIJO2_HIJO1_P1] ] [.HIJO2_P1] [.HIJO3_P1] ]`, no tenemos más que sustituir con cuidado paréntesis y comas, por corchetes y espacios. Pondremos este objeto dentro de un documento de clase `standalone` para que aparezca sólo el árbol. Además, para poder ver de manera resaltada las palabras clave como `gn` o `gadj` entre otras, se añade un diccionario de forma que se pueda personalizar el contenido que aparecerá en su lugar en el árbol. Una vez tenemos el archivo de LaTeX listo, lo escribimos en un fichero y pasamos a la siguiente función.

- `compile`: como ya tenemos nuestro documento listo, debemos compilarlo para poder visualizarlo. Para ello, invocamos al compilador pdfLaTeX con el archivo que `parser` ha escrito (podría haberse optado por cualquier otro compilador), y este nos devolverá un archivo `.pdf` con el árbol listo para visualizar. Borra también archivos temporales.

- `show`: al usuario se le ofrece la opción de recibir el árbol como documento `pdf` o como imagen. Para ello, previamente, ha introducido como argumento `-pdf` o `-img`. Esta función se encarga de, en caso de que se quiera imagen llamar a la función `toJPG` que convierte el `pdf` a imagen `jpg`, y muestra el archivo final mediante el visor de `pdf` o imágenes que el usuario haya definido en su sistema operativo como predeterminado.

Apéndice A

Información relevante

A.1. Requisitos y dependencias

Para la realización de la práctica se han usado una serie de librerías y software externo, por lo que detallamos aquí qué es necesario para poder ejecutarla de manera correcta, y qué versiones se recomiendan.

- **Python**: se ha realizado con la versión `3.8.6`, sin embargo no se descarta que no funcione en versiones superiores. Previamente se deberán instalar una serie de dependencias mediante `pip`:
  - `pip install pyswipl`
  - `pip install unidecode`
  - `pip install shutil`
  - `pip install datetime`
  - `pip install pdf2image`
- **Prolog**: deberá utilizarse **obligatoriamente** la versión `8.4.2` de SWI-Prolog, pues con versiones superiores hemos encontrado incompatibilidades con `pyswipl`.
- **LaTeX**: deberemos tener instalada una distribución de LaTeX en nuestro equipo, en nuestro caso al estar trabajando en Windows hemos optado por MiKTeX (en nuestro caso versión `21.2`), pero en general nos puede servir cualquiera que contenga el compilador pdfLaTeX y que lo tenga añadido como variable de entorno. En caso de no ser esto posible o querer usar otro compilador, no habría ningún problema, pero debería modificarse la línea donde se invoca a este en `compile`. Además, durante la primera ejecución deberemos permitir la instalación de los paquetes necesarios como `qtree`.

A.2. Detalles

Se han cubierto todos los objetivos solicitados, funcionando correctamente las oraciones, sin ningún error de implementación.

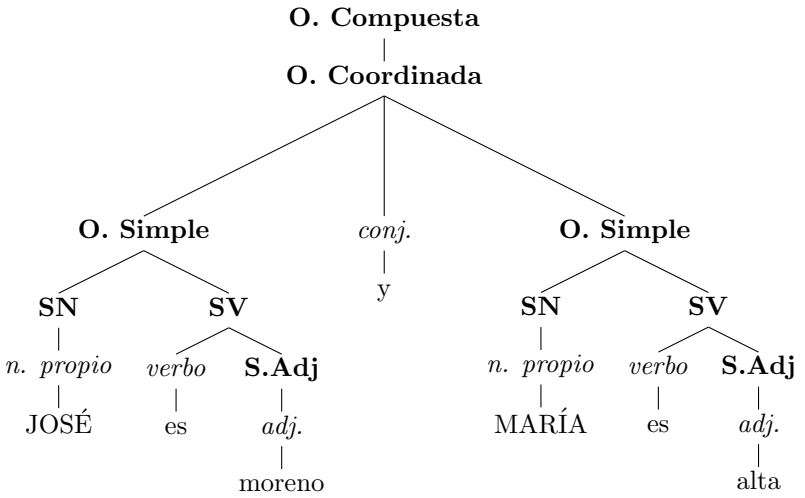
Respecto al reparto de tareas, la práctica se ha realizado por todos los integrantes del grupo a la vez en una llamada de Discord, aportando cada uno soluciones y alternativas a los problemas que se iban encontrando.

Para finalizar, las referencias consultadas han sido, los apuntes de la asignatura, el manual de SWI Prolog, y StackOverFlow.

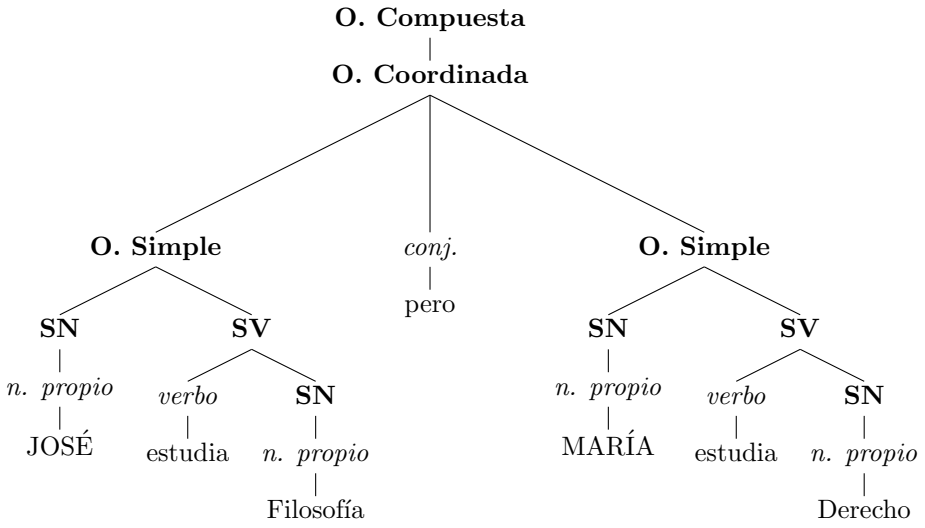
Apéndice B

Resultados análisis

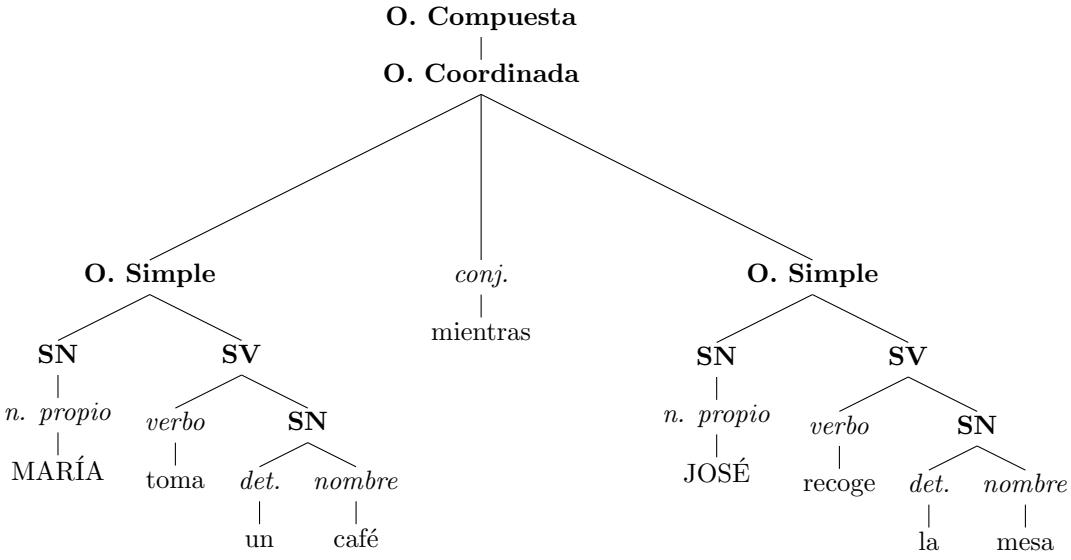
En este apéndice incluimos todos los árboles generados por nuestro programa para las 14 frases solicitadas para la práctica.



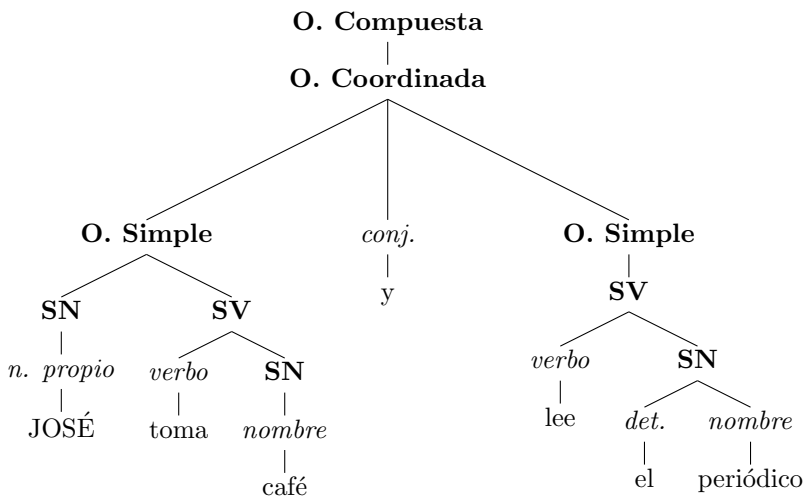

```
-> Traduccion al ingles
JOSE is brunette and MARIA is tall
-> Descomposicion de la oracion compuesta
jose es moreno
maria es alta
```



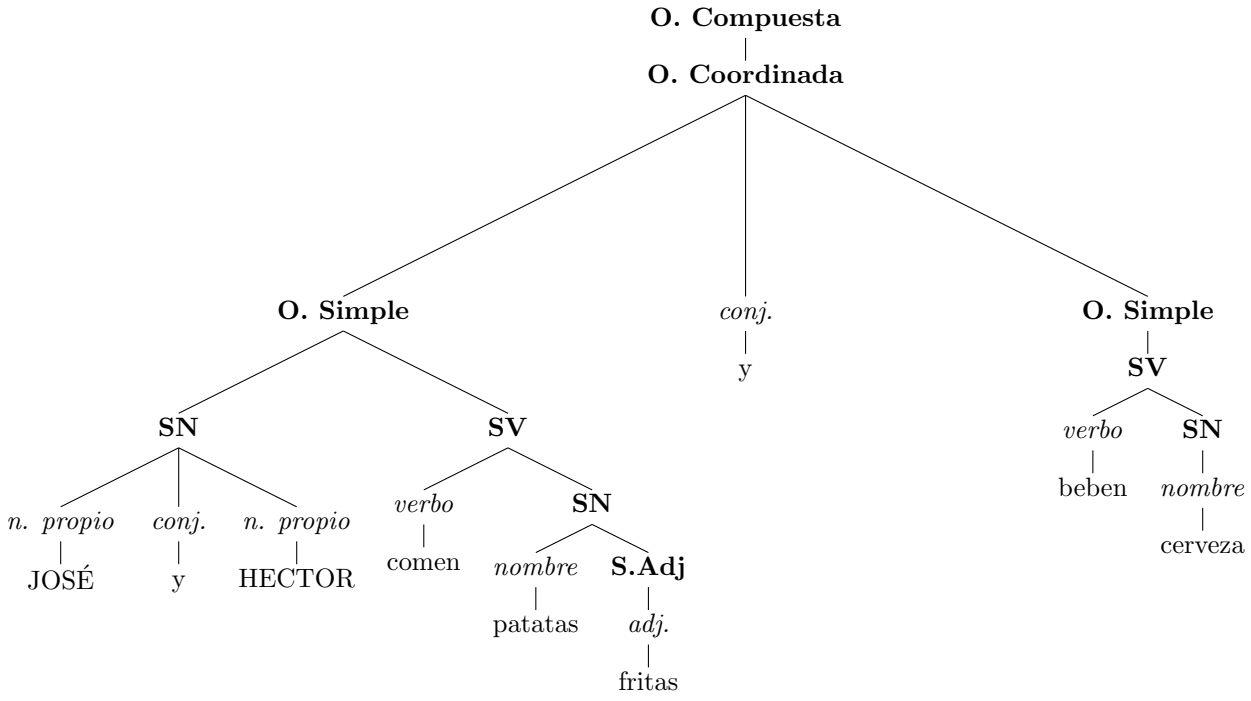

```
-> Traduccion al ingles
JOSE studies Philosophy but MARIA studies Law
-> Descomposicion de la oracion compuesta
jose estudia filosofia
maria estudia derecho
```



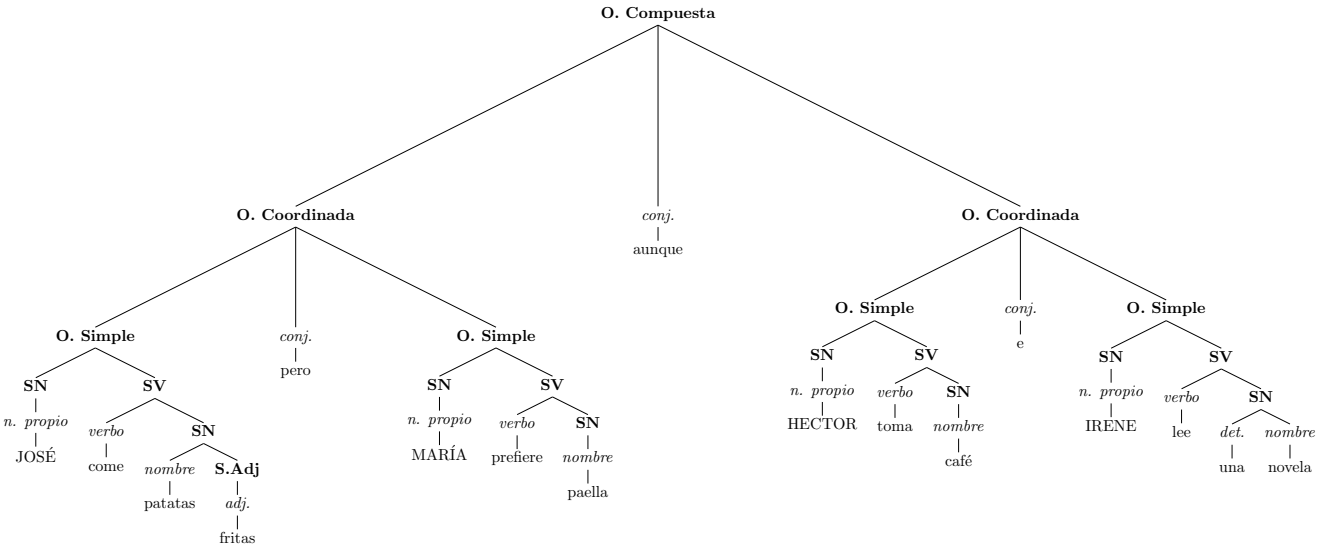

```
-> Traduccion al ingles
MARIA takes a coffee while JOSE cleans up the table
-> Descomposicion de la oracion compuesta
maria toma un cafe
jose recoge la mesa
```



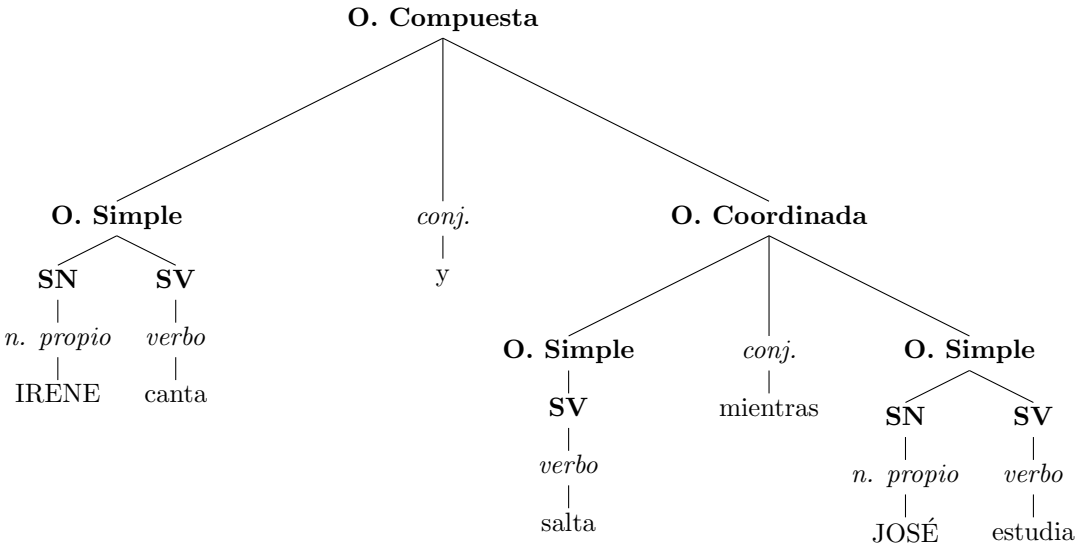

```
-> Traduccion al ingles
JOSE takes coffee and reads the newspaper
-> Descomposicion de la oracion compuesta
jose toma cafe
jose lee el periodico
```



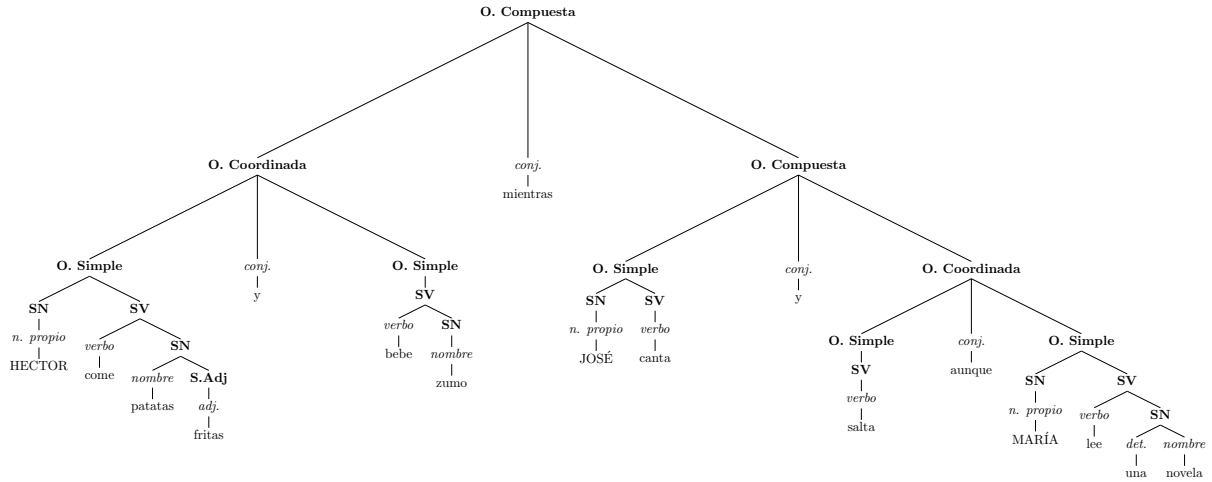

```
-> Traduccion al ingles
JOSE and HECTOR eat fried potatoes and drink beer
-> Descomposicion de la oracion compuesta
jose y hector comen patatas fritas
jose y hector beben cerveza
```



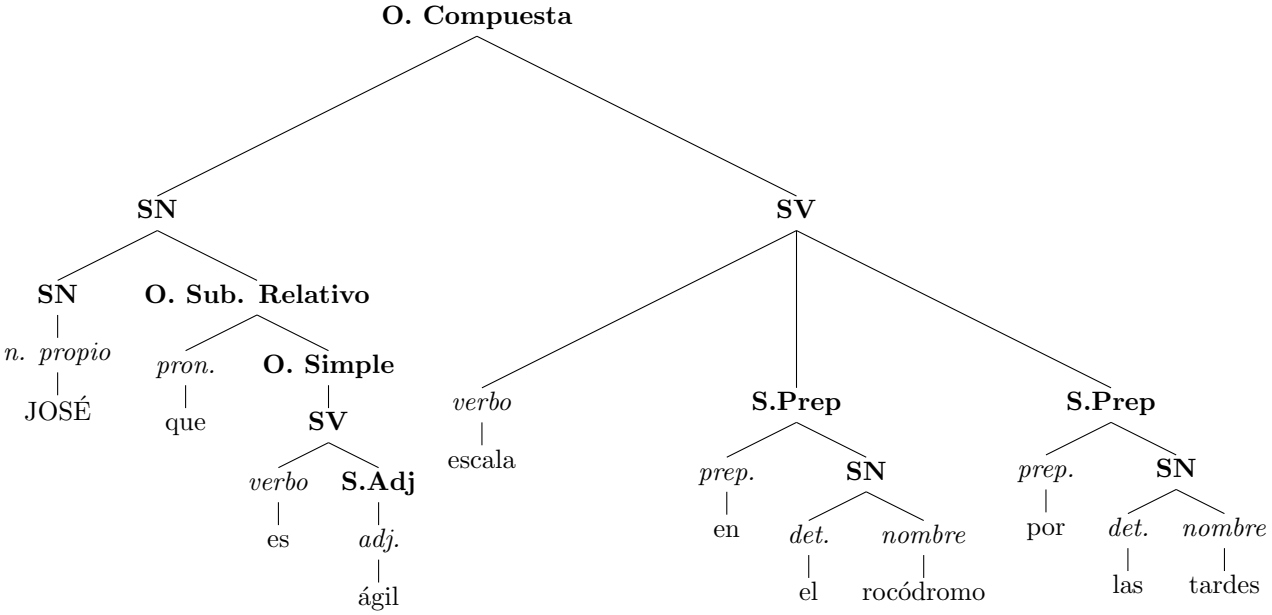

```
-> Traduccion al ingles
JOSE eats fried potatoes but MARIA prefers paella although
HECTOR takes coffee and IRENE reads a novel
-> Descomposicion de la oracion compuesta
jose come patatas fritas
maria prefiere paella
hector toma cafe
irene lee una novela
```



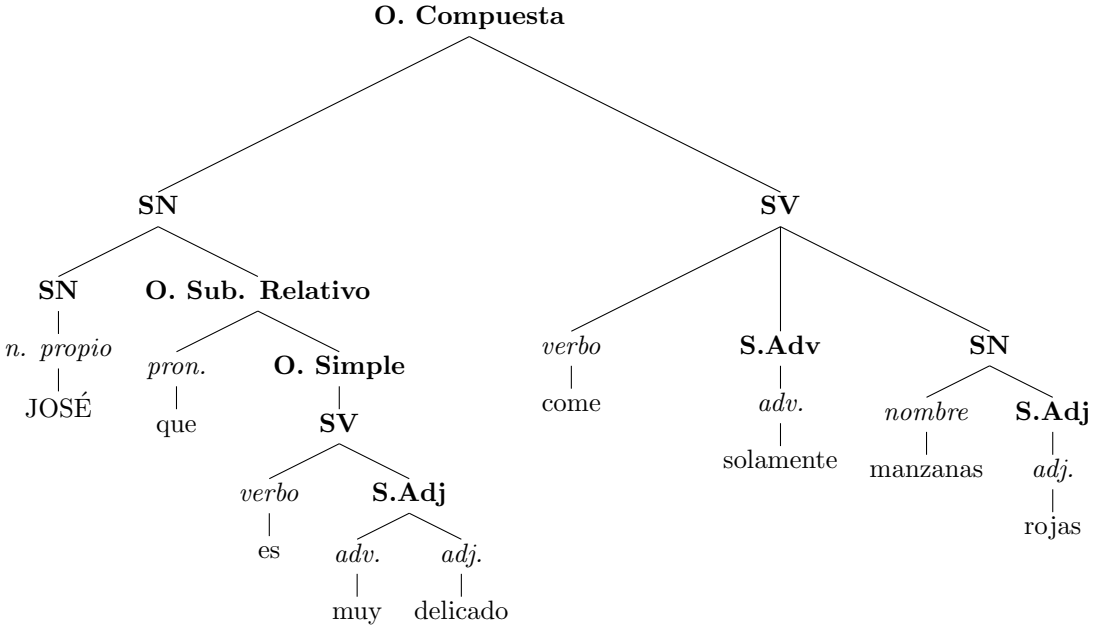

```
-> Traduccion al ingles
IRENE sings and jumps while JOSE studies
-> Descomposicion de la oracion compuesta
irene canta
irene salta
jose estudia
```



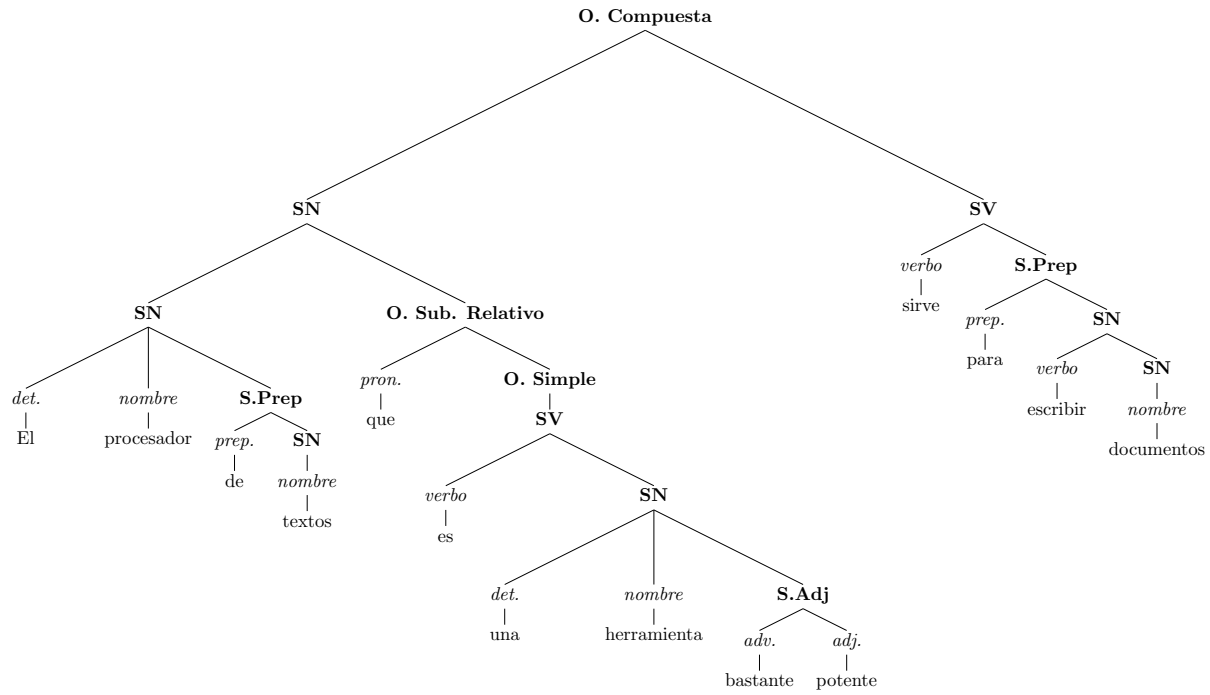

```
-> Traduccion al ingles
HECTOR eats fried potatoes and drinks juice while
JOSE sings and jumps although MARIA reads a novel
-> Descomposicion de la oracion compuesta
hector come patatas fritas
hector bebe zumo
jose canta
jose salta
maria lee una novela
```



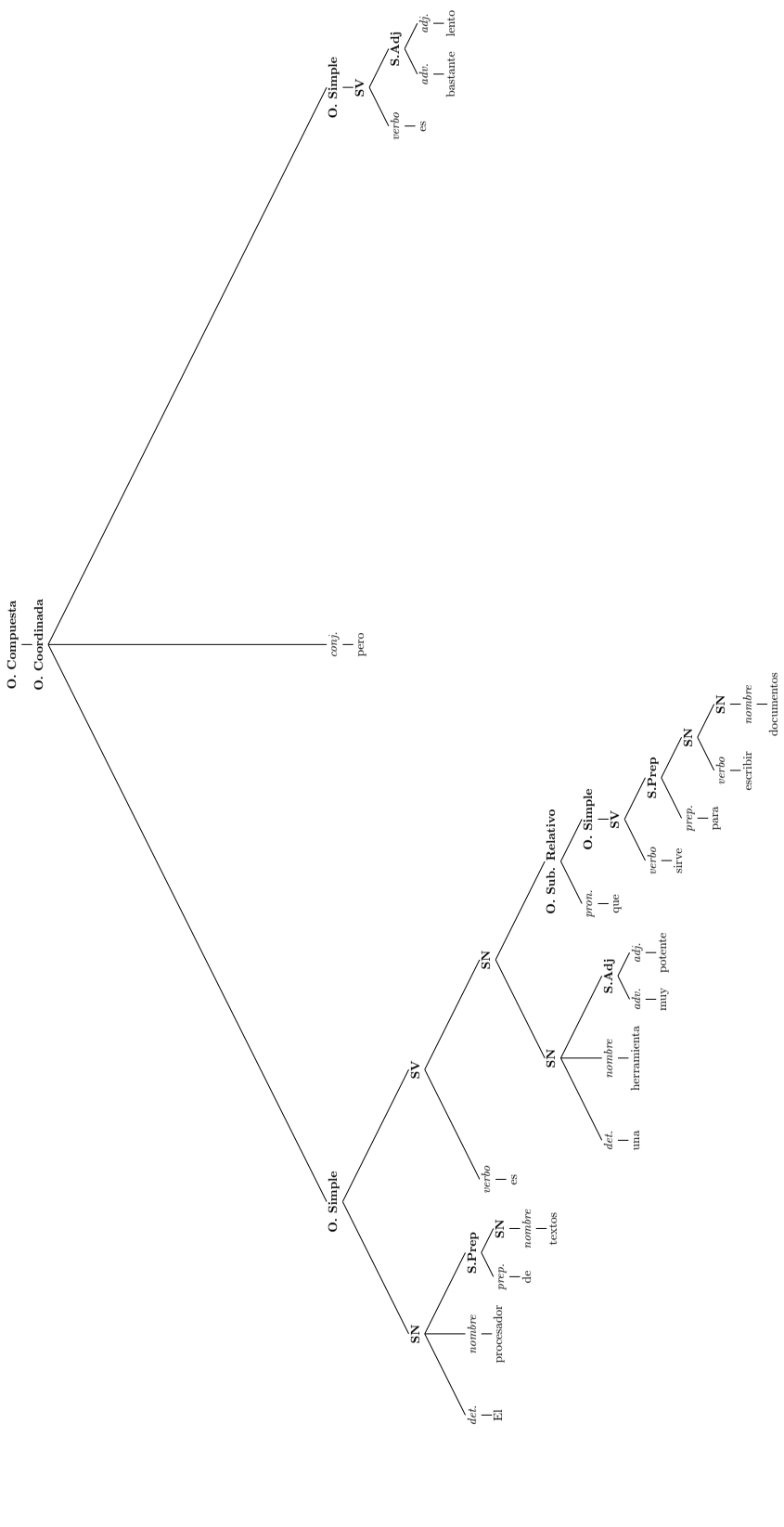

```
-> Traduccion al ingles
JOSE that is agile climbs at the climbing wall in the afternoons
-> Descomposicion de la oracion compuesta
jose es agil
jose escala en el rocodromo por las tardes
```



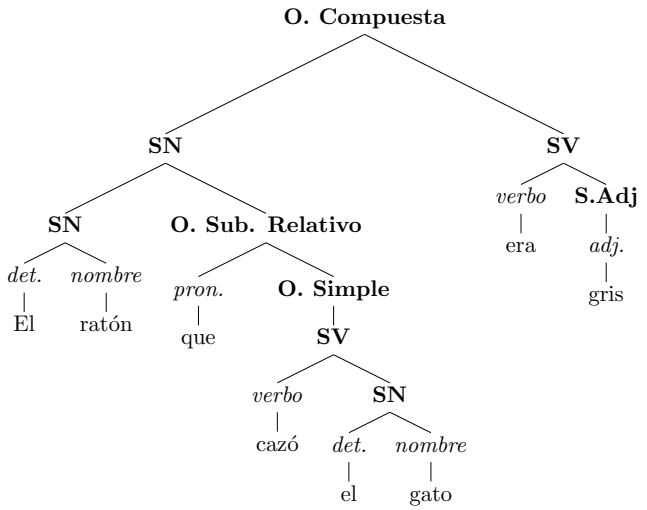

```
-> Traduccion al ingles
JOSE that is very delicate eats only red apples
-> Descomposicion de la oracion compuesta
jose es muy delicado
jose come solamente manzanas rojas
```



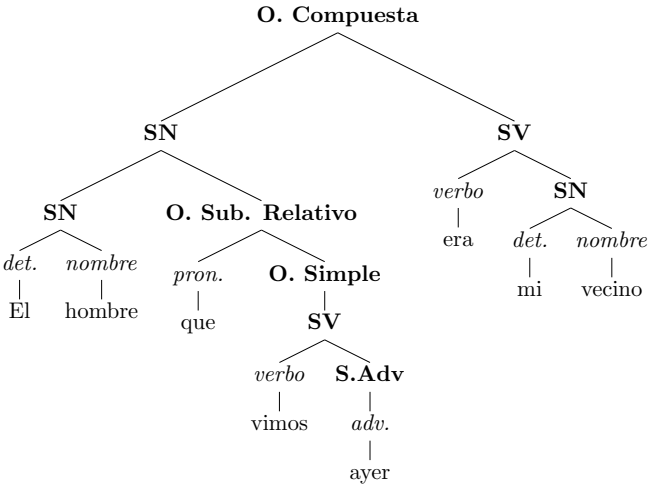

```
-> Traduccion al ingles
The processor of texts that is a quite powerful tool serves for writing documents
-> Descomposicion de la oracion compuesta
el procesador de textos es una herramienta bastante potente
el procesador de textos sirve para escribir documentos
```



-> Traduccion al ingles
The procesador of texts is a very powerful tool that serves for writing documents but is quite slow
-> Descomposicion de la oracion compuesta
el procesador de textos es una herramienta muy potente
el procesador de textos sirve para escribir documentos
el procesador de textos es bastante lento




```
-> Traduccion al ingles
The mouse that was hunted by the cat was grey
-> Descomposicion de la oracion compuesta
el raton cazo el gato
el raton era gris
```




```
-> Traduccion al ingles
The man that we saw yesterday was my neighbour
-> Descomposicion de la oracion compuesta
el hombre vimos ayer
el hombre era mi vecino
```

[0]La descomposición de estas dos últimas oraciones no son del todo semánticamente correctas debido a que las oraciones subordinadas hacen una aclaración del sujeto en vez de complementarlo.