UNIVERSIDAD DE LA REPÚBLICA

Extracción de definiciones y generación automática de crucigramas a partir de textos de prensa

por

Jennifer Esteche

Romina Romero

Tutores:

Luis Chiruzzo, Aiala Rosá

Informe de Proyecto de Grado presentado al Tribunal Evaluador
como requisito de graduación de la carrera
Ingeniería en Computación

en la

Facultad de Ingeniería

Montevideo, Uruguay
Mayo 2015

"If we knew what it was we were doing, it would not be called research, would it?"

Albert Einstein

UNIVERSIDAD DE LA REPÚBLICA

Resumen

Facultad de Ingeniería

Ingeniería en Computación

por Jennifer Esteche
Romina Romero

En este proyecto se plantea el diseño y la implementación de un sistema que tome textos en lenguaje natural en español y genere crucigramas completos (tablero y pistas) a partir de las definiciones contenidas en ellos, de forma totalmente automática. La solución propuesta divide el problema en dos partes: un módulo de extracción de definiciones que emplea *pattern matching* implementado en Python, y otro de generación de crucigramas que utiliza una estrategia *greedy* implementado en Prolog. Se logró conseguir una precisión en las definiciones del 73 % con crucigramas similares a los construidos por humanos.

Palabras clave
procesamiento del lenguaje natural, generación automática de crucigramas, extracción de definiciones

Agradecimientos

Agradecemos a nuestras familias y amigos, por acompañarnos durante todo el camino. También queremos agradecer a nuestros compañeros de carrera, con quienes aprendimos juntos, por todos los buenos momentos. Valoramos los aportes de Williams y Javier para mejorar el informe. Y agradecemos especialmente la guía de nuestros tutores, Luis y Aiala, quienes siempre nos atendieron con buena disposición.

Índice general

Índice de figuras

Índice de tablas

Abreviaciones

PLN **P**rocesamiento del **L**enguaje **N**atural

PoS ***P**art **o**f **S**peach* (categoría gramatical)

GLC ***G**ramática **L**ibre de **C**ontexto*

Dedicamos este trabajo a nuestras familias y amigos.

Capítulo 1

Introducción

Planteo del problema

La extracción de definiciones significa reconocer, dentro de un texto, aquellos fragmentos que están describiendo, de manera más o menos precisa, a otros términos, también presentes en el texto. Estos términos pueden estar compuestos por una o más palabras, habitualmente consecutivas.

El problema de extracción de definiciones está enmarcado en otro más general, el de extracción de información, por lo que los algoritmos desarrollados para el primero pueden ser utilizados como componentes para el segundo. La extracción de información puede describirse como la tarea de recibir textos y devolver ciertos datos allí contenidos, en un formato fijo, sin ambigüedades. Estos datos pueden ser usados para mostrarlos al usuario o pueden ser almacenados en una base de datos, como insumo para otros procesos. Por ejemplo, sirven de insumo para la aplicación de recuperación de información, como son los motores de búsqueda web.

El reconocer dichas relaciones y obtener tanto los términos definidos (definendums) como sus definiciones representa un desafío para el Procesamiento de Lenguaje Natural (PLN), ya que requiere identificar cómo estas relaciones se establecen y determinar qué elementos se pueden abstraer para caracterizar este vínculo. Implica conjugar distintos niveles de análisis de textos, desde los morfológicos hasta los semánticos, pudiendo requerirse conocimiento del mundo. El lenguaje es una entidad que muta con el tiempo y tiene una estructura muy variable de un texto a otro, por lo que su análisis y modelado para un problema que busca relaciones nunca va a encontrar un algoritmo infalible.

Las definiciones que se buscan en este proyecto están enmarcadas en el objetivo final de generar crucigramas a partir de ellas, por lo cual los definendums no tienen por qué hacer

referencia a objetos, pueden ser eventos, personas, etc. Por ejemplo, una palabra puede ser "Hollande" y su definición sería "presidente de Francia". Este par podría extraerse de una oración como la siguiente: "El presidente de Francia, François Hollande, habló en conferencia de prensa".

Por otra parte, la generación de crucigramas constituye un problema básicamente algorítmico: se busca conseguir una manera eficiente de seleccionar un conjunto de elementos dentro de una lista dada, y disponerlos dentro de un tablero cumpliendo con ciertas restricciones. Estas restricciones implican algunas características básicas y otras de carácter estético. Las características básicas incluyen, por ejemplo, que las palabras deben entrar en el tablero y no se deben formar palabras inexistentes. Las características estéticas son subjetivas y refieren, por ejemplo, a que el crucigrama debe ser agradable e interesante para el usuario, por lo cual no debería tener demasiados casilleros negros ni ser excesivamente fácil o difícil. Se debe, entonces, modelar las características subjetivas como una serie de parámetros objetivos que, junto con las otras imposiciones, constituyen un problema de satisfacción de restricciones.

En particular, el problema tratado en este proyecto implica extraer definiciones a partir de textos de prensa en idioma español y utilizarlas para generar crucigramas de manera totalmente automática. Es importante considerar que la cantidad de definiciones extraídas podría no ser suficiente para cubrir las necesidades del crucigrama a generar, por lo cual se generarán algunos recursos adicionales, para obtener las definiciones faltantes. Observar que las definiciones extraídas van a estar muy vinculadas a los textos de prensa, por lo que para alguien que no los leyó podrían no tener sentido.

Motivación

Los crucigramas constituyen, además de un pasatiempo, una herramienta didáctica que puede ser empleada para enseñar temas concretos, en el caso de crucigramas temáticos, o para desarrollar el vocabulario y dominio del lenguaje, en el caso de crucigramas generales.

La extracción de definiciones, por su parte, constituye una importante tarea en sí misma. Permite facilitar el acceso a información de interés ya sea mediante la interacción directa con un usuario, como funcionando de insumo para otras aplicaciones de extracción de información más complejas.

Considerando el gran crecimiento de los volúmenes de información hoy disponibles, en la web por ejemplo, es clara la necesidad de herramientas que faciliten la búsqueda y

la obtención de información, para que los datos relevantes se puedan encontrar. La extracción de información permite reducir dramáticamente la cantidad de tiempo que una persona gasta leyendo textos para encontrar ciertos datos relevantes, a la vez que constituye un insumo fundamental para otras aplicaciones que requieren grandes volúmenes de información como entrada.

Además, el hecho de trabajar en PLN sobre el idioma español constituye un desafío interesante, ya que se cuenta con pocas herramientas, generalmente de menor precisión, y escasa información, respecto a otros idiomas (especialmente en comparación con el inglés). La tarea de extracción de información no se considera en la actualidad una "tarea resuelta", sino que es objeto de investigación, por lo cual se espera poder realizar aportes en estos aspectos.

Objetivos planteados

Los objetivos de este proyecto son:

1. Extraer términos definidos (definedums) y sus descripciones (definiciones) a partir de textos de prensa, de forma totalmente automática.
2. Generar crucigramas usando la mayor cantidad posible de definiciones extraídas en forma automática, complementadas con definiciones externas en caso de ser necesario.

Resultados esperados

1. Obtener un proceso de extracción de pares $\prec$ *definición* , *definendum* $\succ$ basados en textos en lenguaje natural.
2. Obtener un proceso de generación de crucigramas a partir de un conjunto de definiciones preestablecido.
3. Investigar el estado del arte tanto de la generación de crucigramas como de la extracción de definiciones, y realizar un resumen sucinto de los métodos empleados para lograr ambas tareas.

Organización del documento

El resto del documento está organizado de la siguiente manera:

El capítulo 2 se divide en dos secciones: en la primera se brinda una breve introducción a algunos conceptos que es necesario manejar para el correcto entendimiento del trabajo; en la segunda se ofrece un resumen del estado del arte en cuanto a la extracción de definiciones y la generación automática de crucigramas.

En el capítulo 3 se presenta la solución planteada, de forma global y concisa, explicando la arquitectura del sistema y la función de cada módulo.

En el capítulo 4 se profundiza en el proceso de obtención de definiciones. Se explica detalladamente cómo se extraen las definiciones a partir de los textos, y los mecanismos auxiliares empleados.

En el capítulo 5 se profundiza en la generación de crucigramas, explicando los algoritmos empleados para dicho propósito.

En el capítulo 6 se tratan los detalles de implementación y las herramientas utilizadas.

En el capítulo 7 se presentan las evaluaciones realizadas y se analizan los resultados obtenidos.

En el capítulo 8 se dan las conclusiones y lineamientos de trabajo futuro.

Luego siguen las referencias utilizadas y un glosario de términos empleados durante el informe.

Finalmente se cuenta con tres apéndices. En el apéndice A se brinda información general sobre el área de procesamiento del lenguaje natural, para aquellos lectores que no están tan familiarizados con la disciplina. En el apéndice B se ofrece una explicación más profunda del funcionamiento de una de las herramientas utilizadas durante la extracción de definiciones, las expresiones regulares recursivas. En el apéndice C se explica más en profundidad el primer diseño creado para la generación de crucigramas.

Capítulo 2

Marco de trabajo

2.1. Conceptos necesarios

En esta sección se definirán los conceptos necesarios para una mejor comprensión del trabajo. El lector que desee obtener más detalles podrá referirse al apéndice A o a la bibliografía utilizada.

Para los efectos de nuestro proyecto, manejaremos los conceptos de *definendum*, *definición* y *crucigrama* de la siguiente manera:

Definendum: término que se está definiendo.

Definición: término que describe a un definendum dado.

Ejemplo 2.1. *Definición y definendum*

- *Definendum: Montevideo.*
- *Definición: Capital de Uruguay.*

Crucigrama: Un **crucigrama** consiste en una cuadrícula (**tablero**) en donde cada **casillero** puede estar en blanco (casillero vacío) o en negro (casillero negro), y un conjunto de pistas (también llamadas claves). Las pistas son pequeños textos que tratan de describir las palabras que deberán colocarse en la cuadrícula. Dichas palabras se colocarán en casilleros vacíos contiguos horizontal o verticalmente (slots), insertando un carácter alfabético en cada uno. El crucigrama está completo cuando todos los casilleros en blanco han sido completados.

FIGURA 2.1: Ejemplo de crucigrama

Ejemplo 2.2. *Crucigrama*
En la figura 2.1 [1]*, se aprecia a la izquierda el crucigrama vacío, y a la derecha el crucigrama completo. Se trata de un tablero de* 5×5*, con casilleros negros en las posiciones (3,4) y (4,3) (se indica primero la fila y luego la columna), y casilleros blancos en las demás posiciones, definiéndose así 7 slots horizontales y 7 verticales.*

En el marco de este trabajo, el definendum será una de las palabras que eventualmente deberá ir dentro de un crucigrama, y su definición será la pista. No tiene que ser una definición exacta, pero sí dar una descripción suficiente para un crucigrama. En caso de tratarse de un definendum con más de una palabra, se procesará de forma de elegir una de ellas, mientras que las demás podrán formar parte de la pista.

Ejemplo 2.3. *Definiciones y pistas*
Dado el par $\prec$ *definición , definendum* $\succ$

$$\prec \textit{La presidenta de la Sociedad Uruguaya de Glaucoma , Alicia Martínez} \succ$$

se puede generar una pista para el crucigrama: "(Alicia _____). La presidenta de la Sociedad Uruguaya de Glaucoma" donde la palabra solución es "Martínez".

[1] Tomada de http://www.pasatiemposparallevar.com/como-se-juega/crucigrama/.

Extracción automática de definiciones

La extracción automática de definiciones, concerniente a este trabajo, consiste en obtener un término (definendum) y su definición a partir de un texto de referencia, mediante el uso de un algoritmo que lleve a cabo esta tarea.

En este proyecto se busca que esta actividad sea realizada de forma totalmente automática, por lo cual el algoritmo debe funcionar sin intervención humana.

Generación automática de crucigramas

La generación automática de crucigramas consiste en la construcción de uno o más algoritmos que crean el tablero, asignando los casilleros negros, las palabras y las pistas correspondientes. Como salida se obtiene el tablero vacío y las pistas, mientras que las palabras y su correcta distribución en la cuadrícula constituyen la solución.

2.2. Estado del arte

Para abordar este trabajo se pueden separar dos grandes tópicos: extracción de definiciones y generación de crucigramas.

2.2.1. Extracción de definiciones

Métodos de extracción

En lo que refiere a la extracción de definiciones se observa que la metodología de investigación en general recurre al uso de patrones, y consiste en dos etapas secuenciales:

1. definición y aplicación de patrones
2. aplicación de filtros para eliminar definiciones y/o patrones no deseados

Respecto al primer punto, existen dos vertientes:

1. realizar un primer acercamiento buscando patrones manualmente, y luego automatizarlos para encontrar otras definiciones y/o nuevos patrones (Ortega Mendoza [21], Walter y Pinkal [31])
2. manejar una lista preestablecida de patrones (Przepiórkowski et al. [23])

Con el fin de conseguir grandes corpus [2] para trabajar, se suelen emplear motores de búsqueda web o grandes enciclopedias como Wikipedia (Przepiórkowski et al. [23],Rigutini et al. [25]).

Algunos investigadores utilizan corpus anotados conformados por un conjunto de documentos que cubren diferentes dominios (Del Gaudio y Branco [12], Przepiórkowski et al. [23], Sierra [28]). Otros se basan en textos con cierta estructura predefinida, por ejemplo al utilizar diccionarios, tesauros o enciclopedias la estructura es bastante clara, y en el caso de textos de dominios específicos (Sánchez y Márquez [27], Borg, Rosner y Pace [7], Ranaivo-Malançon et al. [24]) también existen ciertos patrones lingüísticos que se suelen emplear, lo que facilita la extracción de definiciones. En el reconocimiento de patrones, muchas veces se emplean expresiones regulares dada su simplicidad y eficiencia (Przepiórkowski et al. [23], Borg, Rosner y Pace [7], Degórski, Marcińczuk y Przepiórkowski [11]).

Se nota una fuerte coincidencia entre los autores que trabajaron en idiomas distintos al inglés respecto a la falta de herramientas disponibles para dichos idiomas, lo que implica una dificultad adicional (Przepiórkowski et al. [23],Ortega Mendoza [21],Degórski, Marcińczuk y Przepiórkowski [11]).

El trabajo de extracción de definiciones suele tener un componente fuertemente relacionado al idioma sobre el cual se está trabajando, y cómo es su gramática. En particular, el uso de anotaciones morfosintácticas, parsing [3] , y la posterior búsqueda de patrones morfosintácticos parecerían ser las técnicas más empleadas para este tipo de tareas. Sin embargo, el trabajo de Ortega Mendoza [21] plantea un mecanismo que pretende independizarse lo más posible del uso de información sintáctica, limitándose al uso exclusivo de información léxica, lo que llevó a utilizar técnicas de data mining para compensar la pobre generalización lograda con este método, y considerar variaciones de género y número en los patrones. Asimismo, el trabajo de Walter y Pinkal [31] - enfocado en textos jurídicos en alemán -, si bien utiliza reglas de extracción, plantea un enfoque basado en ontologías, explotando así la relación entre los conceptos, y emplea un parser orientado a semántica.

[2] Un *corpus* es una colección de material lingüístico con ciertas características deseadas: material escrito u oral, cantidad de idiomas que presenta (monolingüe vs. multilingüe), tipo de texto que se trata (por ej.: prensa, literario, científico), dominio abarcado (por ej.: arte, lingüística), anotado o no. Un *corpus anotado* es aquel que cuenta con anotaciones que lo enriquecen lingüísticamente. Estas surgen a partir de un proceso de análisis manual o automático. Las anotaciones pueden ser de distintos niveles: morfológicas, sintácticas, léxico-semánticas, entre otros.

[3] Las anotaciones morfosintácticas dan información sobre la morfología de las palabras y su categoría gramatical. El parsing es el análisis sintáctico de las oraciones, es decir, cómo están estructuradas las mismas.

Sierra [29] y Ranaivo-Malançon et al. [24] proponen realizar el parsing utilizando gramáticas de dependencias y considerar el orden relativo de las palabras para determinar cuál es la palabra que se está definiendo y cuáles son los límites de su definición.

Degórski, Marcińczuk y Przepiórkowski [11] proponen en cambio desarrollar una gramática parcial a mano, y aplicarle a continuación un conjunto de clasificadores que deciden por votación cuál es la clasificación correcta (una oración dada es o no es una definición). Esto constituye un enfoque interesante, ya que parece intentar aprovechar lo mejor de ambas técnicas, pero hay que destacar que las pruebas se realizaron sobre un corpus relativamente pequeño, donde los métodos de aprendizaje automático no logran lucir su mayor potencial.

Varios autores realizan una categorización de las definiciones, distinguiendo las que contienen el verbo "ser" como conector, las que se conectan mediante otros verbos distintos de "ser" y las que contienen características de puntuación que separan el término que está definido y la definición misma. Es particularmente interesante que esta clasificación se utiliza en los trabajos de Del Gaudio y Branco [12] y de Borg, Rosner y Pace [7], basándose el primero en textos en portugués y el segundo en textos en inglés. Del Gaudio y Branco [12] proponen desarrollar tres gramáticas regulares, una para cada uno de los tres tipos de definiciones para poder realizar la clasificación y separar así las oraciones que corresponden a una definición de las que no.

En general se emplean arquitecturas basadas en varias capas o etapas, donde pequeños módulos realizan pequeñas tareas, y dichos módulos se conectan, habitualmente de forma secuencial, para lograr procesamientos más complejos.

En Rigutini et al. [25] se plantea un sistema basado en capas, donde la última capa del analizador de PLN asigna a los *chunks* previamente identificados, su función sintáctica (sujeto, predicado nominal, predicado verbal, etc.).

Existe otro enfoque que permite la extracción de definiciones sin realizar un reconocimiento de patrones utilizando algoritmos evolutivos[4]. El trabajo de Borg, Rosner y Pace [7] tiene como objetivo explorar el uso de algoritmos evolutivos, para entrenar clasificadores, que separan las oraciones que contienen definiciones de las que no. Se experimenta con estos métodos para aprender las formas lingüísticas típicas de las definiciones y su importancia relativa. A su vez se utilizan técnicas de machine learning para extraer las definiciones de los textos no técnicos y reducir la participación humana de expertos. Los resultados son muy positivos, muestran la viabilidad de explorar aún más el uso de estas técnicas en la extracción de definiciones. Estos algoritmos son capaces de aprender reglas

[4]Los algoritmos evolutivos son métodos que tienen como objetivo buscar y optimizar soluciones basándose en los principios de la evolución biológica, como la mutación y el cruzamiento.

similares a las que un experto en lingüística utiliza, y clasificar los candidatos a definiciones ordenados según qué tan confiables son las formas lingüísticas que los reconocen. Los autores observan que los diferentes estilos de escritura y definiciones plantean un reto importante para la identificación de las definiciones.

Dificultades encontradas

En base a los textos leídos, se pueden identificar principalmente tres dificultades pertinentes para este proyecto:

1. extracción de definiciones extensas
2. empleo de patrones largos
3. falta de herramientas

El primer punto refiere a la dificultad reportada por Del Gaudio y Branco [12] y Sierra [28] en cuanto a que las definiciones extensas son más difíciles de reconocer, ya sea porque se extienden por varias oraciones, o porque suelen verse "interrumpidas" por tramos de texto que no pertenecen a la definición en sí misma. El segundo punto refiere a que los patrones largos suelen tener baja precisión (Sierra [28]). El tercer punto refiere al problema de las pocas herramientas disponibles para la mayoría de los idiomas y su baja precisión, siendo la excepción el inglés (Przepiórkowski et al. [23], Ortega Mendoza [21],Degórski, Marcińczuk y Przepiórkowski [11]).

Evaluación

En lo que refiere a la evaluación, en algunas ocasiones se utilizan mediciones manuales de tasa de acierto respecto a un corpus con las definiciones anotadas, se trabaja también con inter-annotator agreement [5]. Las principales medidas utilizadas son la precisión, el recall y la medida F - como es el caso de Del Gaudio y Branco [12], Borg, Rosner y Pace [7], Ortega Mendoza [21], Sierra [28], Sánchez y Márquez [27], Harabagiu, Bejan y Morărescu [16], Bach y Badaskar [4] entre otros -. Además Ortega Mendoza [21] plantea un método automático iterativo que estima el valor de confianza de la definición: una tupla <palabra, definición>es más confiable a mayor número de patrones que la extraen, y un patrón se hace más confiable a medida que extrae un mayor número de tuplas confiables, lo que es similar al orden de confianza empleado por Borg, Rosner y Pace [7].

[5] El *inter-annotator agreement* es el porcentaje de coincidencias de las anotaciones realizadas sobre un cierto corpus, por distintos anotadores. Da una idea del nivel de consenso entre dichos expertos sobre cuáles son las etiquetas correctas para dicho corpus.

2.2.2. Generación de crucigramas

La generación de crucigramas es una tarea compleja dado el amplio espacio de búsqueda, que a su vez es disperso. El *branching factor* puede superar rápidamente 10^6 en etapas tempranas de la búsqueda (Rigutini et al. [25]). El problema de generación de crucigramas es NP completo (Engel et al. [13]). La resolución de un crucigrama es un problema de *Constraint Satisfaction Programming*: se tienen restricciones de *layout* (la palabra empieza y termina en un borde o en un casillero negro) y del esquema actual (palabras insertadas previamente) (Rigutini et al. [25]).

Para lograr esta tarea hay diversas técnicas utilizadas, coincidiendo en la estrategia de generar por un lado la cuadrícula y luego realizar la asignación de las palabras (y sus definiciones) a la misma.

En Rigutini et al. [25] el sistema de resolución utiliza una cola de prioridad que almacena los conjuntos de soluciones parciales. En cada paso se desencola una solución parcial y se elige un slot de inserción. Un conjunto de palabras candidatas se obtiene realizando consultas indizadas (es esencial optimizar el índice). Las restricciones de *layout* se usan para determinar el largo de la palabra, y cuando hay letras escritas en el slot se imponen mayores restricciones. Se mantiene un índice *full-constraint* para palabras de largo menor a 10, donde una lista de palabras compatibles se guarda para cada combinación de restricciones. Para palabras más largas, se almacena un índice invertido <letra, posición>, y cuando se imponen múltiples restricciones, se obtiene la lista de palabras que cumple cada restricción usando el índice, y luego se intersectan los conjuntos.

En cada paso se selecciona el slot con mayor cantidad de espacios en blanco para limitar el espacio de búsqueda lo más posible. Se puede utilizar un cierto grado de aleatoriedad para devolver distintos resultados en distintas corridas. Cada vez que se inserta una palabra, se calcula el puntaje del esquema obtenido en base a la cantidad de cuadrantes completados y a qué tan probable es que el esquema sea expansible en futuras iteraciones (beneficio del nuevo término). El beneficio se calcula realizando un paso *look-ahead*, mirando cuántas definiciones serán compatibles en la próxima iteración, con la nueva palabra, en los slots que la cruzan.

En Aherne y Vogel [1] se utilizan predicados Prolog, aprovechando la técnica de backtracking para la asignación de palabras. Los autores no realizan la extracción de las definiciones, sino que las mismas son extraídas del WordNet, obteniendo relaciones entre las palabras para brindar una orientación temática al crucigrama.

En Engel et al. [13] primero se crea la cuadrícula y luego se agregan las palabras mediante backtracking, al igual que Aherne y Vogel [1], pero se utilizan algoritmos evolutivos

para la generación del crucigrama y se extraen las definiciones de un diccionario. Se implementan dos algoritmos evolutivos: uno genético[6] y otro memético[7]. Durante los experimentos con el algoritmo genético, los autores observan que la convergencia es muy lenta. Parecería que las ventajas de los algoritmos genéticos no pueden ser explotadas con un enfoque simple, por lo cual desarrollan un algoritmo memético, obteniendo una mayor convergencia del algoritmo.

[6] En un algoritmo genético se realiza la selección de individuos de forma aleatoria, los operadores (mutación, cruzamiento) se aplican a dichos individuos según una probabilidad y se reemplazan los padres por los hijos, manteniendo al mejor individuo de la población anterior.

[7] Un algoritmo memético puede verse como una variante de uno genético que, a diferencia de estos, incorpora la mayor cantidad posible de conocimiento del dominio durante el proceso de generación de una nueva población.

Capítulo 3

Solución propuesta

La solución propuesta genera el puzzle completo: el tablero del crucigrama, los términos y definiciones a utilizar y su ubicación en el crucigrama.

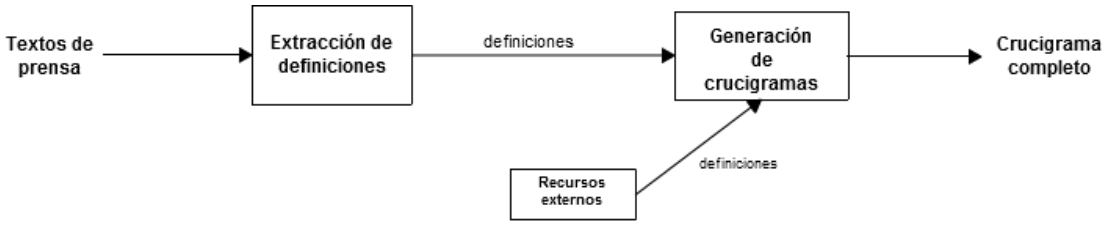

FIGURA 3.1: Diseño de solución propuesta

Como se aprecia en la figura 3.1 el sistema recibe textos de prensa y se los entrega al *módulo de extracción de definiciones*. Este obtiene las definiciones que se encuentran en dichos textos y las devuelve como salida. El *módulo de recursos externos* se trata de un conjunto estático de definiciones adicionales. Luego, el *módulo de generación de crucigramas* recibe todas estas definiciones y con ello genera y devuelve un crucigrama completo.

El motivo por el cual se necesita el *módulo de recursos externos* es que las definiciones extraídas automáticamente podrían no ser suficientes para generar un crucigrama válido o interesante, por lo que este pequeño módulo complementa la salida del de *extracción de definiciones*.

Los textos de entrada corresponden a prensa extraída de medios web, por lo que, antes de poder utilizarlos, es necesario ejecutar un preprocesamiento. Allí se realiza un proceso de "limpieza", donde se descarta el texto que no interesa (como podría ser: links a otros artículos de interés, fecha de edición del artículo, autor, publicidad), y se le da al texto

objetivo un formato adecuado según lo que esperan los módulo que lo consumen (en la sección 6.1 se dan los detalles de esta fase).

En cuanto a la metodología de extracción de definiciones, con base en el estado del arte, se puede decir que existen tres estrategias:

1. sistemas basados en aprendizaje automático
2. sistemas basados en reglas
3. sistemas híbridos

La carencia de textos anotados hizo poco adecuado el uso de métodos de aprendizaje automático supervisado. Utilizar un sistema basado en reglas exclusivamente era viable, pero podría afectar la cantidad de definiciones encontradas. Por ello se optó por un sistema híbrido: crear un pequeño conjunto de reglas manualmente - alrededor de 30 -, e intentar incrementarlo con técnicas de aprendizaje semi-supervisado (*bootstrapping* de patrones).

Para extraer las definiciones contenidas en los textos se obtiene información lingüística y se combinan distintas técnicas que permiten identificar los constituyentes de las oraciones y transformarlos en pares $\prec$ *definición* , *definendum* $\succ$, como se indica en la tabla 3.1.

Información utilizada
información morfosintáctica
información sintáctica
reconocimiento y clasificación de entidades con nombre
Técnicas empleadas
patrones sintácticos
pattern matching
bootstrapping de patrones

TABLA 3.1: Información y técnicas para la extracción de definiciones

Es importante notar que las definiciones que se extienden por más de una oración están fuera del alcance de este proyecto.

En cuanto a la generación del tablero se proponen dos implementaciones: un algoritmo de *backtracking* y un algoritmo *greedy*. Ambos intentan utilizar la mayor cantidad de términos extraídos por el módulo de extracción de definiciones, dejando las otras definiciones como último recurso.

En resumen, el sistema se divide básicamente en dos grandes módulos:

- Módulo de extracción de definiciones

- Entrada: textos de prensa.
- Salida: pares $\prec$ *definición* , *definendum* $\succ$.

▪ Módulo de generación del tablero

- Entrada: pares $\prec$ *definición* , *definendum* $\succ$.
- Salida: un crucigrama completo (con palabras solución y cuadrados negros colocados en el tablero, y las pistas correspondientes a cada palabra).

En los siguientes capítulos se describirán en detalle todos los módulos. A continuación se brinda un ejemplo completo del funcionamiento del sistema.

Ejemplo 3.1. *Funcionamiento del sistema*

Consideremos los siguientes fragmentos de texto pertenecientes a un corpus de textos de prensa:

> *"Esta ocupación justificó la instalación de una base de la Organización del Tratado del Atlántico Norte (OTAN) en el Atlántico Sur".*
> *"Las hojas son las principales consumidoras de agua dentro de la fisiología vegetal".*
> *"Por ejemplo, podemos pensar en Java como un buen lenguaje de programación porque entre otras cosas, es multiplataforma".*
> *"El realizador georgiano afincado en Francia, Otar, visitó en la jornada de ayer algunos de los escenarios que, como Heleta, acogieron el rodaje de su película Euzkadi été 1982".*
> *"Según el estudio, 63 por ciento de los vendedores recibe a sus clientes con saludos poco cordiales, como «hola», «¿sí?» o «¿qué necesita?»".*
> *"Ana Olivera declaró «Visitante Ilustre de Montevideo» al rey de Ijero (Nigeria), Joseph Adebayo Adewole".*
> *"Un campesino contó su historia: 'yo aré durante 30 días esta tierra, la sembré, y ahora las inundaciones estropearon todo' ".*
> *"Ante una infección evite compartir utensilios y tazas, lave los platos con agua caliente y jabonosa, y evite fumar en los alrededores de la vivienda".*

A partir de ellas el módulo de extracción de definiciones obtiene los siguientes pares $\prec$ *definición , definendum* $\succ$ *:*

$\prec$ *Organización del Tratado del Atlántico Norte , OTAN* $\succ$

$\prec$ *principales consumidoras de agua dentro de la fisiología vegetal , hojas* $\succ$

$\prec$ *buen lenguaje de programación , Java* $\succ$

$\prec$ *realizador georgiano afincado en Francia , Otar* $\succ$

≺ *saludo poco cordial , hola* ≻

≺ *(_ Olivera) declaró «Visitante Ilustre de Montevideo» al rey de Ijero (Nigeria) , Ana* ≻

Por su parte, el módulo de recursos externos ofrece, entre otras, las siguientes definiciones:

≺ *voz del verbo 'arar' , aré* ≻

≺ *voz del verbo 'lavar' , lave* ≻

Luego, el módulo de generación de crucigramas, con todas las definiciones obtenidas, construye el crucigrama de la figura 3.2

FIGURA 3.2: Crucigrama autogenerado

Capítulo 4

Obtención de definiciones

En las siguientes secciones se explicará detalladamente la obtención de las definiciones: extracción de definiciones y recursos externos.

4.1. Extracción de definiciones

Para extraer las definiciones partimos de un conjunto de patrones de base, el cual podría incrementarse mediante la técnica de bootstrapping de patrones, similar a lo descrito en Hearst [17][1], permitiendo así encontrar más definiciones mediante la aplicación de pattern matching.

La construcción de los patrones de base la realizamos de forma manual, mediante la observación de textos de prensa en español de medios web - similares a los de los corpus a utilizar -, y mediante la adopción de algunos patrones sugeridos en Ortega Mendoza [21]. Dichos patrones se mejoraron en sucesivas iteraciones considerando los resultados sobre el corpus de desarrollo.

En el alcance propuesto para este proyecto, solo buscamos obtener pares $\prec$ *definición* , *definendum* $\succ$ que estén contenidos dentro de la misma oración.

Para poder realizar el pattern matching sobre el texto, necesitamos someter el corpus a una serie de pasos (ver figura 4.1), que básicamente consisten en:

- Análisis morfosintáctico (PoS tagging)
- Segmentación en proposiciones
- Parsing

[1] Por más datos ver Apéndice A.

Una proposición es una unidad lingüística de estructura oracional, esto es, constituida por sujeto y predicado, que se une mediante coordinación o subordinación a otra u otras proposiciones para formar una oración compuesta.

A partir de las definiciones encontradas buscamos nuevos patrones, mediante la técnica de *bootstrapping*.

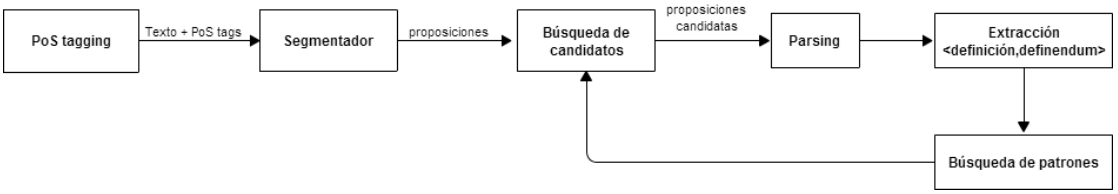

FIGURA 4.1: Diseño del módulo de extracción

Los patrones utilizados constan de dos partes, el patrón léxico y el patrón parser:

$$Patrón = <pLexico,pParser>$$

El *patrón léxico* funciona como un filtro para obtener candidatos a definiciones en base al texto, mientras que el *patrón parser* utiliza el árbol de parsing de dichos candidatos para obtener las definiciones o descartar los candidatos fallidos.

La necesidad de contar con estos dos componentes surge porque el parser utilizado no analiza correctamente las oraciones largas. Para evitar extraer pares $\prec$ *definición* , *definendum* $\succ$ erróneos, limitamos el texto a analizar a la proposición más chica que contiene al candidato.

Primero, realizamos un análisis del texto para conocer la morfología y la categoría gramatical de cada palabra, utilizando un PoS tagger[2]. Este paso es requisito para el parsing, además de aportar información potencialmente útil sobre las palabras[3].

Luego, procedemos a segmentar cada oración en proposiciones para así quedarnos con aquellas que contienen candidatos a definiciones, utilizando el *patrón léxico*. Estas últimas se analizan sintácticamente con un parser basado en gramáticas libres de contexto. A partir de los árboles de parsing "exitosos" extraemos los pares $\prec$ *definición* , *definendum* $\succ$ con la ayuda del *patrón parser*.

Por último, aplicamos la técnica de *bootstrapping* para encontrar nuevos patrones y definiciones. Cabe mencionar que realizamos un post procesamiento de las definiciones para brindarles un formato adecuado para utilizarlas en los crucigramas.

[2] (*Part-Of-Speech Tagger*) software que lee texto en algún idioma y asigna etiquetas gramaticales (*PoS tags*) a cada palabra (y otros tokens), como por ejemplo *nombre, verbo, adjetivo*, etc..

[3] Luego de realizar algunas pruebas, observamos que los mejores resultados se obtienen generalizando los patrones en cuanto a las características morfológicas de las palabras, por lo cual la información de género y número no es utilizada.

El pseudocódigo en **Algoritmo 1** muestra el algoritmo de extracción.

Algoritmo 1 Algoritmo completo de extracción

Entrada: texto, patrones
Salida: definiciones, patrones

```
patrones = ... % Se inicializan los patrones de base, patrón = ⟨pLexico,pParser⟩
definiciones = []
texto + PoS_tags = pos_tagger(texto)
proposiciones = segmentar_en_proposiciones(texto + PoS_tags)
Para cada patrón en patrones :
    % Búsqueda de candidatos: se buscan las coincidencias entre el texto (etiquetado gramaticalmente) y su patrón léxico
    candidatos = pattern_matching(texto + PoS_tags, pLexico)
    Para cada candidato en candidatos :
        % se busca la proposición más chica que lo contiene íntegramente
        propMin = get_proposicion_minima(candidato, proposiciones)
        % se analiza sintácticamente dicha proposición
        arbol_parsing = parsing(propMin)
        % se buscan coincidencias entre el parsing y el patrón parser.
        coincidencias = pattern_matching(arbol_parsing, pParser)
        Para cada coincidencia en coincidencias :
            % se extraen la definición y el definendum, según indique el pParser
            < definicion, definendum >= extraer_def(coincidencia, pParser)
            % se agrega el par al conjunto de definiciones luego de post procesarlas para darles formato para crucigramas
            definiciones+ = aplicar_formato(< definicion, definendum >)
            % Buscar patrones: se buscan otras apariciones del par <definición,definendum>en el texto para encontrar nuevos patrones
            nuevos_patrones = buscar_patrones(< definicion, definendum >, texto + PoS_tags)
            patrones = agregar_patrones(patrones, nuevos_patrones)
        fin para
    fin para
fin para
```

Consideremos la siguiente oración:

"Un periodista de Subrayado afirmó que Tabaré Vázquez, quien es el actual Presidente de la República, decidió no hacer comentarios sobre el incidente."

Al aplicarle el PoS tagger, obtenemos el texto etiquetado que se muestra en la tabla 4.1.

Palabra	Lema	Categoría gramatical
Un	uno	Determinante
periodista	periodista	Nombre
de	de	Preposición
Subrayado	subrayado	Nombre
afirmó	afirmar	Verbo
que	que	Conjunción

Continúa en la siguiente página...

Palabra	Lema	Categoría gramatical
Tabaré_Vázquez	tabaré_vázquez	Nombre Propio
,	,	Puntuación
quien	quien	Pronombre
es	ser	Verbo
el	el	Determinante
actual	actual	Adjetivo
Presidente_de_la_República	presidente_de_la_república	Nombre Propio
,	,	Puntuación
decidió	decidir	Verbo
no	no	Adverbio
hacer	hacer	Verbo
comentarios	comentario	Nombre
sobre	sobre	Preposición
el	el	Determinante
incidente	incidente	Nombre
.	.	Puntuación

TABLA 4.1: Texto con etiquetas gramaticales

Luego, lo separamos en proposiciones:

[*Un periodista de Subrayado afirmó*
[*que Tabaré Vázquez ,*
[*quien es el actual Presidente de la República*]
, decidió
[*no hacer comentarios sobre el incidente*]
]
]

En este caso, el *patrón léxico* encuentra una única proposición candidata: *" que Tabaré Vázquez, quien es el actual Presidente de la República, decidió no hacer comentarios sobre el incidente"*

A continuación, hallamos el árbol de parsing que se muestra en la figura 4.2.

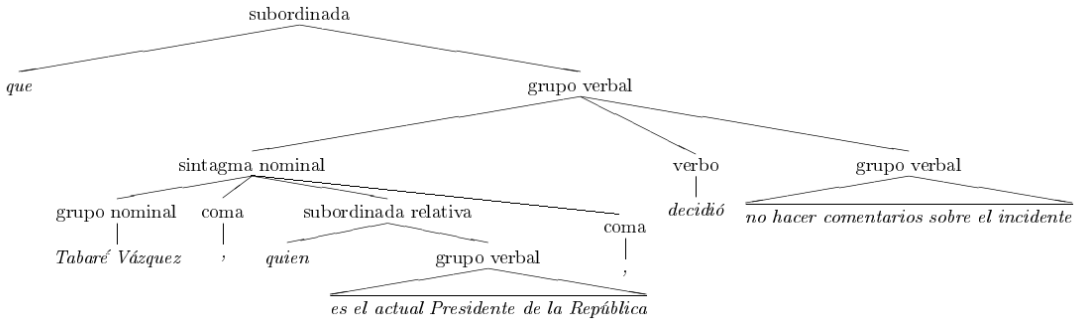

FIGURA 4.2: Árbol de parsing

Finalmente, utilizamos el *patrón parser* para extraer el par

$$\prec \textit{Quien es el actual Presidente de la República , Tabaré Vázquez} \succ$$

y, a partir de este, buscamos nuevos patrones aplicando *bootstrapping*.

4.1.1. Diseño del patrón

a. Patrón léxico

Este componente del patrón consiste en una lista de tuplas $\langle palabra, lema, tag \rangle$, donde cada elemento coincide con una o más palabras en el texto.

Los componentes *palabra* y *lema* corresponden a los determinados por el PoS tagger y pueden estar subespecificados, en cuyo caso se aceptará cualquier palabra o lema respectivamente. El tercer componente de esta tupla, *tag*, contiene una lista de etiquetas gramaticales que serán aceptadas (o rechazadas) y la cantidad máxima de palabras a aceptar (rechazar). Esto último lo representamos por medio de los siguientes atributos:

- **Lista de tags**: lista de etiquetas gramaticales.
- **Acepta**: indica si se deben rechazar o aceptar las palabras cuyos tags se encuentren en la *lista de tags*.
- **Max**: cantidad máxima de palabras de la lista a ser aceptadas o rechazadas según *acepta*.

Max se determinó heurísticamente para cada patrón, luego de analizar manualmente las definiciones en diversos textos.

Cabe mencionar, que los atributos palabra y lema pueden hacer referencia a un signo de puntuación (como ser coma, punto, guión, etc.).

Para la extracción de definiciones es necesario conocer la categoría gramatical de las palabras y la clasificación semántica de las entidades con nombre (*persona, lugar, organización u otro*).

El elemento ⟨**palabra: P, lema: L, lista de tags: LT, acepta: A, max: M**⟩ se interpreta según sus valores, como se indica en la siguiente tabla:

⟨ P, L, {LT, TRUE, 1}⟩	acepta la palabra **P**, con lema **L** y cuya categoría está en **LT**
⟨ P, L, {LT, FALSE, 1}⟩	acepta la palabra **P**, con lema **L** y cuya categoría **NO** está en **LT**
⟨_, _, {LT, TRUE, M}⟩ con M >1	acepta hasta **M** palabras cuyas categorías están en **LT**
⟨_, _, {LT, FALSE, M}⟩ con M >1	acepta hasta **M** palabras cuyas categorías **NO** están en **LT**

Tabla 4.2: Interpretación de los elementos del patrón léxico

Decimos que una secuencia de palabras coincide con el *patrón léxico* si cada palabra de dicha secuencia coincide con un elemento del patrón. A su vez, una palabra coincide con un elemento del patrón si es aceptada por este.

Este diseño permite tener flexibilidad para generalizar los patrones, de forma de abarcar una gran cantidad de definiciones que cumplen con cierta estructura (la descrita en el patrón). Asimismo, permite tener elementos más específicos (por los campos *palabra*, y en menor medida *lema*), para evitar capturar falsos candidatos.

Si se tiene el texto "*Leucemia Linfocítica Crónica (LLC)*"dentro de un corpus, y se quiere extraer el siguiente par: ≺ *Leucemia Linfocítica Crónica* , *LLC* ≻ , un patrón para detectarlo se puede construir como se muestra en la tabla:

Palabra	**Patrón léxico** ⟨ **palabra**: P, **lema**: L,{**lista de tags**: LT, **acepta**: A, **max**: M}⟩
Leucemia_Linfocítica_Crónica	⟨ _, _, {[Nombre], TRUE, 1}⟩
(	⟨(, (, {[Paréntesis], TRUE, 1}⟩
LLC	⟨_, _, {[Nombre], TRUE, 1}⟩
)	⟨),), {[Paréntesis], TRUE, 1}⟩

Tabla 4.3: Construcción del patrón léxico

b. Patrón parser

El segundo componente del patrón consiste en una lista de triplas de la forma ⟨*parámetro1,parámetro2,parámetro3*⟩, donde el valor de los últimos dos parámetros varía

según el valor del primero, como se indica en la tabla 4.4.

Parámetro1	**Parámetro2**	**Parámetro3**
≺DEF≻ (definición)	lista de tags que debe contener o excluir la definición	tipo de constituyente que corresponde a la definición
≺DFM≻ (definendum)	lista de tags que debe contener o excluir el definendum	tipo de constituyente que corresponde al definendum
≺DEFC≻ (complemento de definición)	-	tipo de constituyente que corresponde al complemento de la definición
≺DFMC≻ (complemento de definendum)	-	tipo de constituyente que corresponde al complemento del definendum
palabra o no especificado	lema o no especificado	categoría gramatical

TABLA 4.4: Parámetros del patrón parser

En los casos que la *definición* (*definendum*) presenta, según el análisis sintáctico, un constituyente principal y complementos, empleamos ≺DEFC≻ (≺DFMC≻) para representar estos últimos.

El último tipo de elemento (*parámetro1 = palabra o no especificado*) representa las palabras que actúan como nexo entre la *definición* y su *definendum*. Estas serán utilizadas posteriormente, en el post procesamiento de las definiciones (ver sección 4.1.4), para completar la *definición*[4].

La tabla 4.5 muestra las abreviaciones de los principales tipos de constituyente utilizados.

Constituyente	**Abreviación**
Sintagma nominal	sn
Sintagma adverbial	sadv
Sintagma preposicional	sp
Sintagma adjetival	s-a
Grupo	grup
Grupo verbal	grup-verb
Grupo nominal	grup-nom
Grupo preposicional	grup-sp
Coordinación nominal	coor-n
Coordinación verbal	coor-vb

TABLA 4.5: Abreviaciones de los principales constituyentes

[4]En esta instancia no forman parte de la definición, ya que no se utilizan para la búsqueda automática de nuevos patrones.

Dado el patrón parser

$$\text{pParser} = [\langle \prec \text{DFM} \succ, [Nombre], sn \rangle, \langle \_, ser, Verbo \rangle, \langle \prec \text{DEF} \succ, \_, sn, \rangle]$$

Patrón parser 4.1.1

podemos extraer la siguiente información:

- El *definendum* está constituido por un sintagma nominal[5] (*sn*).
- El *definendum* debe contener un *Nombre*.
- La *definición* está constituida por un sintagma nominal (*sn*).
- Ambos están relacionados a través del verbo *ser*.
- La secuencia de palabras que contiene al par $\prec$ *definición* , *definendum* $\succ$ es de la forma *sintagma nominal + verbo ser + sintagma nominal*.

Consideremos el texto "*Al parecer, el hombre es un animal racional*", cuyo árbol de parsing se presenta en la figura 4.3.

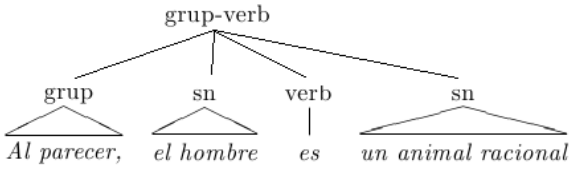

FIGURA 4.3: Árbol de parsing: "*Al parecer, el hombre es una animal racional*"

Para obtener el par $\prec$ *un animal racional* , *el hombre* $\succ$, aplicamos el *patrón parser* propuesto. En la figura 4.4 se muestra la correspondencia entre los constituyentes sintácticos del texto y los elementos del patrón.

[5]Sintagma que tiene como núcleo o elemento principal un nombre o sustantivo. En la frase "aquel día lluvioso volvimos a casa", 'aquel día lluvioso' es un sintagma nominal.

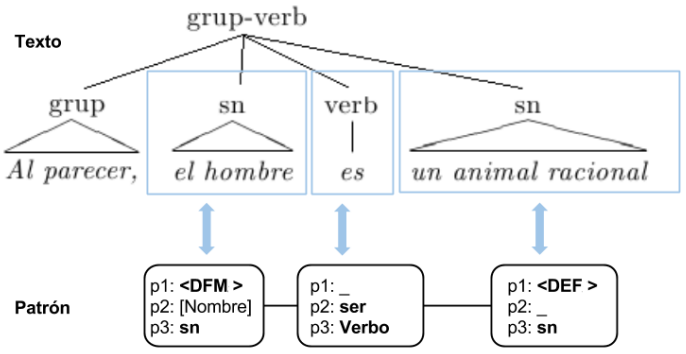

FIGURA 4.4: Correspondencia entre el texto y el *patrón parser*.

Por otro lado, consideremos el texto "*el glaucoma es la segunda causa de ceguera en el mundo*", cuyo árbol de parsing se muestra en la figura 4.5.

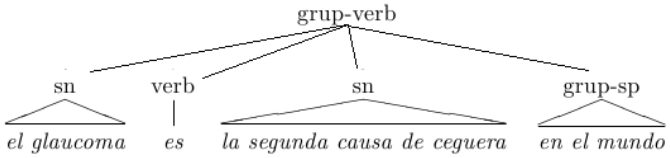

FIGURA 4.5: Ejemplo de constituyente complementario

En este caso, se desea extraer el par $\prec$ *la segunda causa de ceguera en el mundo* , *el glaucoma* $\succ$, por lo que, para obtener la definición completa, debemos capturar tanto el sintagma nominal "*la segunda causa de ceguera*", como el grupo preposicional "*en el mundo*". Para esto, empleamos $\prec$DEFC$\succ$ para representar a este último grupo, modificando el *patrón parser* mostrado en 4.1.1 de la siguiente manera:

$$[\langle \prec \text{DFM} \succ, [Nombre], sn \rangle, \langle \_, ser, Verbo \rangle, \langle \prec \text{DEF} \succ, \_, sn, \rangle, \langle \prec \text{DEFC} \succ, \_, grup\text{-}sp \rangle]$$

Reconocimiento del árbol de parsing

En esta etapa utilizamos el *patrón parser* y expresiones regulares compatibles con Perl (Perl Compatible Regular Expressions) [39]. Considerando la naturaleza del problema, las expresiones regulares tradicionales no son de utilidad, ya que no permiten la utilización de recursión, necesaria para el reconocimiento del par $\prec$ *definición* , *definendum* $\succ$. Utilizamos entonces PCRE, que permiten reconocer cualquier lenguaje libre de contexto [47].

Retomemos el mismo texto de la figura 4.3, el árbol de parsing más detallado puede apreciarse en la siguiente figura:

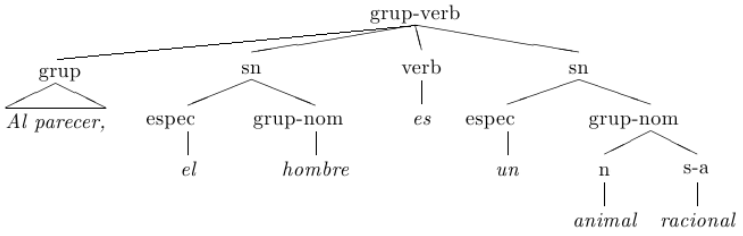

FIGURA 4.6: Árbol de parsing completo: "*El hombre es un animal racional*"

En la salida del parser utilizado, el árbol de la figura 4.6 se representa mediante el anidamiento de los constituyentes sintácticos, donde cada uno de ellos está delimitado por paréntesis rectos, como se puede apreciar en 4.7.

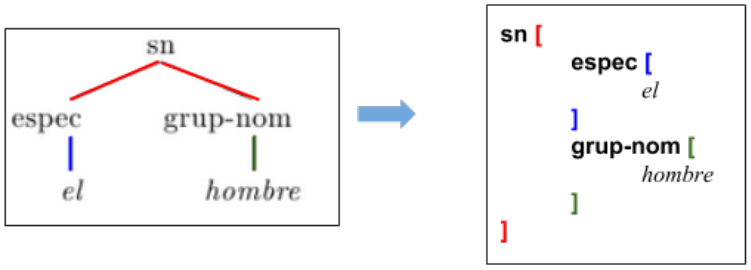

FIGURA 4.7: Ejemplo de salida del parser

Según el patrón parser mostrado en 4.1.1, la expresión debe capturar el primer sintagma nominal como *definendum* y el último como *definición*, teniendo en cuenta que están relacionados por medio del verbo "*ser*". El problema de reconocer los constituyentes no es sencillo, ya que tanto la *definición* como el *definendum* pueden contener varios grupos en su interior (anidando paréntesis rectos), como se aprecia en el ejemplo. Se hace evidente, entonces, la necesidad de utilizar un mecanismo recursivo como PCRE, para reconocer los constituyentes de la oración, y así extraer la *definición* y su *definendum*.

Por más información sobre la construcción de las expresiones utilizadas, referirse al apéndice B.

c. Principales patrones generados

A continuación, se muestran esquemáticamente los principales patrones de base construidos manualmente.

Patrones conectados con “como”:

Patrón léxico ⟨palabra, lema, lista de tags, acepta, max⟩	**Patrón parser** ⟨parámetro1, parámetro2, parámetro3⟩	**Ejemplo**
[⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_, _, {[Nombre], FALSE,5}⟩, ⟨como, como, {[Conjunción], TRUE,1}⟩, ⟨_, _, {[Nombre],FALSE,5}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩]	[⟨≺DEF≻, [Nombre], sn⟩, ⟨como,como,Conjunción⟩, ⟨≺DFM≻, [Nombre], coor-n|sn⟩]	*...especies carismáticas como los grandes felinos, los simios y los elefantes.*
[⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_, _, {[Punto], FALSE,5}⟩, ⟨_, _, {[Coma], TRUE,1}⟩, ⟨como, como, {[Conjunción], TRUE,1}⟩, ⟨_, _, {[Nombre],FALSE,5}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩]	[⟨≺DEF≻, [Nombre], sn⟩, ⟨_,_,Coma⟩, ⟨como,como,Conjunción⟩, ⟨≺DFM≻, [Nombre], sn⟩]	*...otros cultivos, como el fríjol de primera cosecha...*
[⟨_, _, {[Nombre], TRUE,1}⟩, ⟨conocido, conocer, {[Verbo], TRUE,1}⟩, ⟨como, como, {[Conjunción], TRUE,1}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩]	[⟨≺DEF≻, _, sn⟩, ⟨conocido,conocer,Verbo⟩, ⟨como,como,Conjunción⟩, ⟨≺DFM≻, _, sn⟩]	*...la obtención del biocombustible conocido como etanol...*

TABLA 4.6: Patrones generados conectados con “como”

Patrones basados en verbos:

Patrón léxico ⟨palabra, lema, lista de tags, acepta, max⟩	**Patrón parser** ⟨parámetro1, parámetro2, parámetro3⟩	**Ejemplo**
[⟨_, _, {[Determinante], TRUE,1}⟩, ⟨_, _, {[Nombre], FALSE,5}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_, _, {[Punto], FALSE,5}⟩, ⟨_, ser, {[Verbo], TRUE,1}⟩, ⟨_, _, {[Nombre], FALSE,5}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩]	[⟨≺DFM≻,[Nombre Propio], sn⟩, ⟨≺DFMC≻, _, grup-sp⟩, ⟨_, ser, Verbo⟩, ⟨≺DEF≻, [Nombre], sn⟩, ⟨≺DEFC≻, _, grup-sp⟩]	*El glaucoma es la segunda causa de ceguera en el mundo.*

Continúa en la siguiente página...

Patrón léxico	Patrón parser	Ejemplo
[⟨_, _, {[Nombre Propio], TRUE,1}⟩, ⟨_, _, {[Nombre], FALSE,5}⟩, ⟨_, _, {[Verbo], TRUE,1}⟩, ⟨_, _, {[Nombre], FALSE,5}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩]	[⟨≺DFM≻,[Nombre Propio], sn⟩, ⟨≺DFMC≻, _, grup⟩, ⟨_,_,Verbo⟩, ⟨≺DEF≻, _, coor-n|sn⟩, ⟨≺DEFC≻, _, grup|sn⟩]	*Mujica respaldó importante inversión minera.*

TABLA 4.7: Patrones generados basados en verbos

Patrones basados en puntuación:

Patrón léxico ⟨palabra, lema, lista de tags, acepta, max⟩	Patrón parser ⟨parámetro1, parámetro2, parámetro3⟩	Ejemplo
[⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Nombre], FALSE,5}⟩, ⟨(, (, {[Paréntesis], TRUE,1}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩, ⟨),), {[Paréntesis], TRUE,1}⟩]	[⟨≺DEF≻, _, sn⟩, ⟨(, (, Paréntesis⟩, ⟨≺DFM≻,[Nombre propio],grup-nom⟩, ⟨),), Paréntesis⟩]	*Ministerio de Salud Pública (MSP) ...*
[⟨_, _, {[Determinante], TRUE,1}⟩, ⟨_, _, {[Nombre], FALSE,5}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_, _, {[Punto], FALSE,5}⟩, ⟨_, _, {[Coma], TRUE,1}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_, _, {[Coma], TRUE,1}⟩]	[⟨≺DEF≻, _, coor-n|sn⟩, ⟨≺DEFC≻, _, grup-sp⟩, ⟨≺DFM≻,[Nombre Propio],grup-nom⟩, ⟨_, _, Coma⟩]	*El secretario general del Sunca, Oscar Andrade, ...*
[⟨_,_, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Punto], FALSE,5}⟩, ⟨'-','-',{[Guión], TRUE,1}⟩, ⟨_,_, {[Nombre], FALSE,5}⟩, ⟨_,_, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Punto], FALSE,5}⟩, ⟨'-','-',{[Guión], TRUE,1}⟩]	[⟨≺DFM≻, [Nombre], grup-nom⟩, ⟨'–','–',Guión⟩, ⟨≺DEF≻, [Nombre], sn⟩, ⟨'-','-',Guión⟩]	*Puntigliano -ex presidente de la Administración Nacional de Puertos- contó que...*

Continúa en la siguiente página...

Patrón léxico	Patrón parser	Ejemplo
[⟨_,_, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Punto], FALSE,5}⟩, ⟨'–','–',{[Guión], TRUE,1}⟩, ⟨_,_, {[Nombre], FALSE,5}⟩, ⟨_,_, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Punto], FALSE,5}⟩, ⟨'–','–',{[Guión], TRUE,1}⟩]	[⟨≺DEF≻, [Nombre], sn⟩, ⟨'–','–',Guión⟩, ⟨≺DFM≻, [Nombre], sn⟩, ⟨'–','–',Guión⟩]	*Por primera vez una especie marina –el atún rojo– tendrá el papel protagonista.*
[⟨_,_, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Punto], FALSE,5}⟩, ⟨_,_,{[Coma], TRUE,1}⟩, ⟨_,_, {[Nombre], FALSE,5}⟩, ⟨_,_, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Punto], FALSE,5}⟩, ⟨_,_,{[Punto], TRUE,1}⟩]	[⟨≺DEF≻, [Determinante], sn⟩, ⟨≺DEFC≻, _, grup-sp⟩, ⟨_,_,Coma⟩, ⟨≺DFM≻, [Nombre Propio], sn⟩, ⟨≺DFMC≻, _, grup-sp⟩, ⟨_,_,Punto⟩]	*... el presidente de la República, Tabaré Vázquez.*
[⟨_,_, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Punto], FALSE,5}⟩, ⟨':', ':',{[Puntuación], TRUE,1}⟩, ⟨_,_, {[Nombre], FALSE,5}⟩, ⟨_,_, {[Nombre], TRUE,1}⟩]	[⟨≺DEF≻, [Nombre], sn⟩, ⟨≺DEFC≻, _, grup-sp⟩, ⟨':', ':',Puntuación⟩, ⟨≺DFM≻, [Nombre], coor-n|sn⟩]	*...cuatro productos destinados sobre todo al mercado externo: maíz, soja, café y caña de azúcar*[6].

TABLA 4.8: Patrones generados basados en puntuación

4.1.2. Búsqueda automática de patrones

En esta etapa utilizamos el texto taggeado por el PoS tagger, la *definición* y el *definendum*, y buscamos nuevos patrones por medio de una expresión regular.

Denominaremos *def** a la secuencia de lemas de la *definición* y *dfm** a la secuencia de lemas del *definendum*. Entonces, buscamos coincidencias en el texto del estilo

$$def* + secuencia\_conexión + dfm* \text{ o}$$
$$dfm* + secuencia\_conexión + def*$$

Es decir, obtenemos todas las palabras que conectan a *def** y *dfm** en cada aparición del par en el texto. Cabe mencionar que se tienen en cuenta solamente las apariciones de *def* y *dfm** cercanas en el texto (a no más de 5 palabras de distancia).

[6] Recordamos que las definiciones extraídas se post procesan para que contengan una sola palabra (ver sección 4.1.4).

El *patrón léxico* está dado por la secuencia de los lemas y categorías gramaticales de las palabras de *secuencia_ conexión*, mientras que el *patrón parser* es la concatenación de dicha secuencia con la información del tipo de constituyente de la *definición* y el *definendum* dada por el patrón original.

Tomamos la decisión de solo tener en cuenta las palabras que se encuentran entre *def** y *dfm**, ya que las pruebas realizadas para determinar qué palabras anteriores o posteriores forman parte del patrón arrojaron muchos errores. El hecho de solo considerar los lemas y las categorías de la definición y su definendum amplía la cantidad de coincidencias que se encuentran en el texto y, por lo tanto, obtenemos una cantidad mayor de posibles patrones.

Por otro lado, para construir el nuevo patrón, decidimos no tener en cuenta las palabras de *secuencia_ conexión* debido a que limita de forma innecesaria la búsqueda de nuevas definiciones. Sin embargo, sí consideramos los lemas, ya que la categoría gramatical no aporta suficiente información sobre el patrón, lo que generaría la extracción de definiciones erróneas. Existen otras generalizaciones que se pueden realizar para construir el patrón, pero por razones de tiempo no fue posible implementarlas.

Consideremos los siguientes fragmentos de texto pertenecientes a un corpus:

"*El hombre es un animal racional.*", "*El hombre es conocido como un animal racional.*"

y el patrón

Patrón léxico	**Patrón parser**
⟨palabra, lema, lista de tags, acepta, max⟩	⟨parámetro1, parámetro2, parámetro3⟩
[⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_,ser, {[Verbo], TRUE,1}⟩, ⟨_, _, {[Determinante], TRUE,1}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩]	[⟨≺DFM≻, _, sn⟩, ⟨_,ser,Verbo⟩, ⟨≺DEF≻,_,sn⟩]

A partir del *patrón léxico* extraemos el siguiente par ≺ *definición* , *definendum* ≻ :

≺ *un animal racional* , *el hombre* ≻

De este modo, la expresión regular generada para el reconocimiento de nuevos patrones tiene la forma:

[uno,animal,racional] + *secuencia_ conexión* + *[el,hombre]* *o*
[el,hombre] + *secuencia_ conexión* + *[uno,animal,racional]*

Dicha expresión reconoce el texto "*El hombre es conocido como un animal racional*", siendo secuencia_conexión igual a [⟨es, ser, {Verbo,Singular,Presente}⟩, ⟨conocido, conocer, {Verbo,Masculino,Singular}⟩, ⟨como, como, {Conjunción,Subordinada}⟩].

Luego, el patrón nuevo es:

Patrón léxico ⟨palabra, lema, lista de tags, acepta, max⟩	**Patrón parser** ⟨parámetro1, parámetro2, parámetro3⟩
[⟨_,ser, {[Verbo], TRUE,1}⟩, ⟨_,conocer, {[Verbo], TRUE,1}⟩, ⟨_, como, {[Conjunción], TRUE,1}⟩,]	[⟨≺DFM≻, _, sn⟩, ⟨_,ser,Verbo⟩, ⟨_,conocer,Verbo⟩, ⟨_, como,Conjunción⟩, ⟨≺DEF≻,_,sn⟩]

4.1.3. Ejemplo completo de extracción

Consideremos el siguiente patrón:

Patrón léxico ⟨palabra, lema, lista de tags, acepta, max⟩	**Patrón parser** ⟨parámetro1, parámetro2, parámetro3⟩
[⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Nombre], FALSE,5}⟩, ⟨(, (, {[Paréntesis], TRUE,1}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩, ⟨),), {[Paréntesis], TRUE,1}⟩]	[⟨≺DEF≻, _, sn⟩, ⟨(, (, Paréntesis⟩, ⟨≺DFM≻,[Nombre propio],grup-nom⟩, ⟨),), Paréntesis⟩]

Supongamos que dentro de nuestro corpus tenemos los siguientes fragmentos de texto: "*La Administración de Servicios de Salud del Estado (ASSE) es el prestador estatal de salud pública en Uruguay.*" y "*La Administración de los Servicios de Salud del Estado, conocido por su acrónimo ASSE, cuenta con una red de servicios en todo el país.*".

Al buscar las coincidencias del patrón léxico en el texto, obtenemos el siguiente candidato a definición: "*Administración de Servicios de Salud del Estado (ASSE)*". En la figura 4.8 se puede ver la correspondencia palabra-elemento del *patrón léxico*.

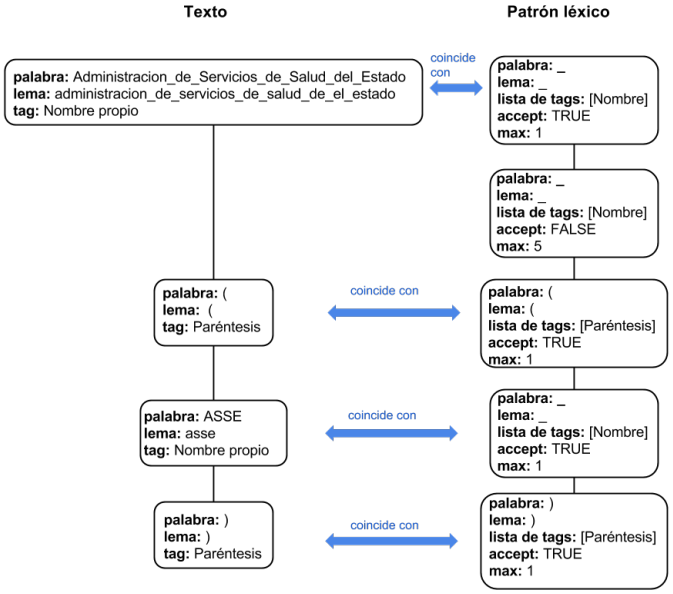

FIGURA 4.8: Correspondencia entre el texto y el *patrón léxico*

Cabe notar que el elemento del patrón ⟨_,_,{[Nombre],FALSE,5}⟩ en este caso no se corresponde con ninguna palabra, pero igualmente el patrón coincide con el texto ya que el elemento indica que debe haber un máximo de 5 palabras de categoría distinta de *Nombre* y en su defecto ninguna.

Luego, hallamos la proposición más chica que la contiene:

"*La Administración de Servicios de Salud del Estado (ASSE) es el prestador estatal de salud pública en Uruguay.*"

y obtenemos el árbol de parsing correspondiente a dicha proposición.

A continuación, con la ayuda del patrón parser, extraemos el siguiente par ≺ *definición* , *definendum* ≻ :

≺ *La Administración de Servicios de Salud del Estado* , *ASSE* ≻ .

Las coincidencias entre el texto y el patrón parser se muestran en la figura 4.9.

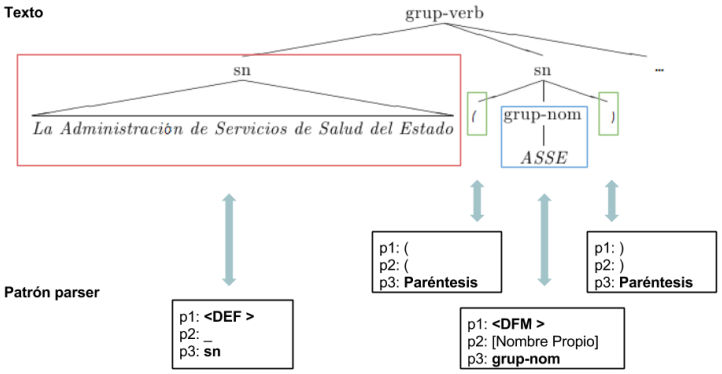

FIGURA 4.9: Correspondencia entre el texto y el *patrón parser*

Finalmente, buscamos nuevas apariciones de la *definición* y su *definendum* cercanas en el texto, obteniendo el fragmento "*La Administración de los Servicios de Salud del Estado, conocido por su acrónimo ASSE, ...*", y construimos el nuevo patrón como se indica en la tabla:

Patrón léxico	**Patrón parser**
⟨palabra, lema, lista de tags, acepta, max⟩	⟨parámetro1, parámetro2, parámetro3⟩
	[⟨≺DEF≻, _, sn⟩,
[⟨_, _, {[Coma], TRUE,1}⟩,	⟨_, _,Coma⟩,
⟨_,conocer, {[Verbo], TRUE,1}⟩,	⟨_,conocer,Verbo⟩,
⟨_, por, {[Preposición], TRUE,1}⟩,	⟨_, por,Preposición⟩,
⟨_, su, {[Determinante], TRUE,1}⟩,	⟨_, su,Determinante⟩,
⟨_, acrónimo, {[Nombre], TRUE,1}⟩]	⟨_, acrónimo,Nombre⟩,
	⟨≺DFM≻,[Nombre propio],grup-nom⟩]

4.1.4. Post procesamiento de las definiciones

Una vez obtenidos todos los pares ≺ *definición* , *definendum* ≻ de los textos de prensa, procedemos a darles un formato adecuado a los mismos para así poderlos utilizar como pistas en el crucigrama.

Consideremos que en la etapa anterior extrajimos el siguiente par:

≺ *José Mujica* , *Uruguay* ≻

Es claro que dicha definición no es adecuada como pista para el crucigrama, ya que "*José Mujica*" no describe a "*Uruguay*". Queremos, entonces, transformar el par anterior en

≺ *País al que pertenece José Mujica* , *Uruguay* ≻

para así poder utilizarlo.

Para obtener las palabras y su correspondiente pista con un formato adecuado para el crucigrama, le aplicamos al par inicial una serie de transformaciones que explicaremos a continuación.

En primer lugar, extraemos la *definición* y el *definendum* completos según el patrón. Esto es, completar la *definición* (*definendum*) con el texto capturado por otros elementos del *patrón parser* (*parámetro1 = palabra o no especificado*).

Si tenemos el par ≺ *importante inversión minera* , *Mujica* ≻ del cual sabemos que está relacionado a través del verbo "*respaldó*", la *definición* y *definendum* resultantes son ≺ *respaldó importante inversión minera* , *Mujica* ≻ .

En segunda instancia, descartamos todas las definiciones que necesiten del contexto para ser entendidas, es decir, aquellas que hacen referencia a una persona, organización, etc., mencionada previamente en el texto y que no se menciona explícitamente en la *definición*. Para esto, buscamos tanto en la *definición* como en el *definendum* pronombres y determinantes que sean posesivos o demostrativos (como ser 'esa', 'este', 'nuestro', 'aquella', 'suya', etc.) y en caso de contenerlos, el par es descartado. Es importante mencionar que la idea original era sustituir ese pronombre o determinante por la entidad a la que hace referencia, pero no encontramos una herramienta que resolviera las correferencias con éxito.

Luego, descartamos también las definiciones cuyo *definendum* solo contiene palabras muy frecuentes[7] en los corpus utilizados, ya que podrían no resultar tan atractivos para completar el crucigrama.

A continuación, completamos la *definición* y su *definendum* con templates. Para esto, utilizamos la clasificación de las entidades con nombre: para cada par ≺ *definición* , *definendum* ≻ , hallamos el nombre principal de cada uno y obtenemos su clasificación semántica (*persona, organización, lugar u otro*). De esta manera, completamos la pista según la relación semántica entre la *definición* y su *definendum*.

Consideremos el siguiente par ≺ *definición* , *definendum* ≻ :

≺ *Ricardo Erlich* , *Montevideo* ≻

La clasificación semántica para "*Ricardo Erlich*" es *Persona* y para "*Montevideo*" es *Lugar*. Como la relación entre la *definición* y el *definendum* es *Persona-Lugar*, completamos la definición con el template "*Lugar al que pertenece*", quedando así el par:

≺ *Lugar al que pertenece Ricardo Erlich* , *Montevideo* ≻

[7] La obtención de las palabras frecuentes se explica en el capítulo 6.

Para algunos patrones, completamos la *definición* teniendo en cuenta solamente la clasificación semántica del *definendum*.

Dado el par $\prec$ *respaldó importante inversión minera* , *Mujica* $\succ$, como la clasificación semántica del *definendum* es *Persona*, la *definición* se completa con "*Quien*", formando el par $\prec$ *Quien respaldó importante inversión minera* , *Mujica* $\succ$.

Para otros patrones, como por ejemplo las siglas, no se utilizan templates.

Por último, a partir de la *definición* y el *definendum* obtenemos los pares $\prec$ *pista* , *palabra* $\succ$ listos para utilizarse en el crucigrama. Notemos que de cada par $\prec$ *definición* , *definendum* $\succ$ pueden generarse varios pares $\prec$ *pista* , *palabra* $\succ$. Para construir dichos pares, obtenemos todos los nombres no frecuentes que contiene el *definendum* y, en caso de que uno sea compuesto, lo separamos en nombres simples.

Consideremos el definendum "*Presidente José_ Mujica*". Los nombres a considerar para el crucigrama son: "*Presidente*", "*José*" y "*Mujica*".

De esta forma, cada uno de los nombres obtenidos pasa a ser una *palabra* a considerar para completar el crucigrama. La pista asociada a cada palabra p se genera a partir de la *definición* anterior y el *definendum*, sustituyendo p por "_____", si el mismo está compuesto por más de una palabra. Si el *definendum* se constituye de una sola palabra, el par $\prec$ *definición* , *definendum* $\succ$ es igual al par $\prec$ *pista* , *palabra* $\succ$.

Dado el par $\prec$ *La presidenta de la Sociedad Uruguaya de Glaucoma* , *Alicia Martínez* $\succ$, obtenemos los pares
$\prec$ *(_____ Martínez). La presidenta de la Sociedad Uruguaya de Glaucoma* , *Alicia* $\succ$ y
$\prec$ *(Alicia _____). La presidenta de la Sociedad Uruguaya de Glaucoma* , *Martínez* $\succ$.

Teniendo en cuenta que estas palabras van a completar un crucigrama, identificamos aquellos pares $\prec$ *pista* , *palabra* $\succ$ que provienen del mismo par $\prec$ *definición* , *definendum* $\succ$, para poder controlar que en la solución no aparezcan dos palabras con la misma definición. Es decir, que no se utilicen en el mismo crucigrama los dos pares generados del ejemplo anterior.

De esta manera, obtenemos todas las palabras y pistas asociadas que serán utilizadas en la etapa de generación de crucigramas para completar el tablero.

4.2. Recursos externos

El módulo de recursos externos ofrece una lista de pares $\prec$ *pista* , *palabra* $\succ$, para ampliar la oferta de palabras disponibles y así poder ofrecer crucigramas más interesantes

para el lector (reduciendo la cantidad de cuadrados negros, en el caso de que las definiciones extraídas no alcanzaran).

Para confeccionar dicha lista se tomaron dos fuentes:

1. WordNet
2. listas de palabras cortas obtenidas a partir de páginas web

WordNet aporta una mayor amplitud léxica, en comparación con los textos de prensa, y definiciones correctas y concretas, por lo cual constituye un recurso interesante para un crucigrama. Dentro de WordNet recopilamos todas las definiciones asociadas a cada uno de los lemas que aparecían, y almacenamos los pares ≺definición,lema≻ en un archivo. Decidimos tomar solamente los lemas y no todas las palabras porque estos ya aportan 13868 definiciones, una cantidad más que suficiente.

Por su parte, la lista de palabras cortas es de gran utilidad para rellenar "pequeños huecos" que se suelen generar al ir construyendo los crucigramas. A partir de un listado de palabras de la Academia Nacional de Letras [44] obtuvimos 67 palabras de 3 y 4 letras. Esta fuente es particularmente interesante porque se trata de un listado de palabras del español del Uruguay con sus respectivas definiciones. Por otro lado, utilizamos un listado de 559 palabras de 2 y 3 letras brindado por la Asociación Uruguaya de Scrabble [46], cuyas definiciones buscamos manualmente dentro de diccionarios, eligiendo la acepción más adecuada en cada caso, o colocando breves descripciones del tipo "*voz del verbo X*", para muchos de los verbos conjugados. Almacenamos los 629 pares ≺definición, palabra corta≻ en un nuevo archivo.

Este proceso de obtención de definiciones a partir de recursos externos se realizó una única vez, generando así dos archivos de recursos propios. Para poder aprovechar las definiciones de este módulo basta con consultar dichos archivos.

Capítulo 5

Generación de crucigramas

Realizamos dos diseños distintos para cumplir con la tarea de generar crucigramas, basando cada uno en una técnica distinta. Inicialmente estaba previsto utilizar *backtracking*, lo que funcionó correctamente pero de forma muy ineficiente, por lo que realizamos adicionalmente otro diseño utilizando *greedy*.

Ambos diseños emplean la misma representación del tablero, y están implementados en un lenguaje de programación lógica, ya que provee mecanismos de unificación [1] eficiente (orden 1) y backtracking implícito.

El tablero lo representamos mediante dos estructuras de matriz. Una de las matrices consiste en la lista de filas del tablero, mientras que la otra es la lista de columnas, lo que permite un acceso más eficiente a las filas o columnas según sea necesario. Si contáramos solamente con la matriz por filas, al querer obtener la columna de la posición k (o un slot vertical) deberíamos recorrer k posiciones dentro de cada una de las filas, en cambio agregando la matriz por columnas, obtenemos su k-ésimo elemento. Manejamos tableros cuadrados (misma cantidad de filas que de columnas), ya que es lo usual en crucigramas. Cada celda tiene inicialmente una variable sin instanciar, que es la misma tanto en la "vista por filas", como en la "vista por columnas", lo que permite que, al asignarle una letra, toda la representación quede automáticamente actualizada, gracias a la unificación. En cada slot deberá colocarse una palabra de los pares $\prec$ *pista* , *palabra* $\succ$ disponibles. La forma en que recorremos los slots y elegimos las palabras a asignar determina el costo

[1] Los *términos* en programación lógica pueden ser constantes (átomos como *mia* y *vicente* o números como *27*, *9*), variables (como *X*, *Y*), o términos complejos (como una lista *[A11,A12,A13]*).
Decimos que dos términos *unifican* si son idénticos o si contienen variables que pueden ser instanciadas uniformemente de forma que el resultado final son términos iguales. El mecanismo de unificación chequea si dos términos unifican, y de ser así realiza las sustituciones necesarias para que queden unificados. En el caso de un slot de largo 3 que contiene una "o" en su segundo lugar y en el resto variables: *[A11,o,A13]*, y la palabra *sol*, representada como una lista de términos atómicos: *[s,o,l]* , vemos que este slot y esta palabra unifican, al instanciar las variables del siguiente modo: $A11 = s$, $A13 = l$. Para más información sobre unificación, referirse al apéndice A.

de encontrar una solución y su calidad (en cuanto a cantidad de casilleros negros, por ejemplo).

Representamos cada palabra como la lista ordenada de sus letras. Podemos asignar una palabra a un slot solo si es compatible con él, y si no fue insertada previamente en el tablero. Decimos que una palabra es compatible con un slot si unifica con el mismo, es decir si todas sus letras son compatibles con los casilleros de este. Una letra es compatible con un casillero si está blanco o tiene la misma letra asignada. Las palabras a asignar las tomamos de los pares $\prec$ *pista* , *palabra* $\succ$ de las fuentes de definiciones, con el siguiente orden de preferencia decreciente:

1. *módulo de extracción de definiciones*
2. *módulo de recursos externos*

Este orden de preferencia se fundamenta en que el objetivo principal es construir crucigramas automáticos a partir de textos, lo que incluye la extracción de definiciones. A pesar de que en este proyecto se prevé un módulo de recursos externos para complementar las definiciones disponibles, el objetivo futuro sería extraer más definiciones e independizarse del módulo auxiliar.

Un crucigrama es válido si no se generan palabras inexistentes en ningún slot. Construimos los slots de manera dinámica, a medida que se agregan casilleros negros. Ambos algoritmos mantienen listas de slots verticales y horizontales, que se van actualizando a medida que se generan nuevos slots.

Ejemplo 5.1. *Representación del tablero*
En la figura 5.1 se muestra la representación de un tablero de 4×4*, con su matriz de filas a la izquierda y su matriz de columnas a la derecha. Cada uno de los Aij representa una variable sin instanciar. En este tablero tenemos 6 slots horizontales y 5 verticales, siendo la lista de slots horizontales: [[A11, A12, A13, A14], [A21], [A23, A24], [A31, A32, A33, A34], [A41, A42], [A44]], y la de slots verticales: [[A11, A21, A31, A41], [A12], [A32, A42], [A13, A23, A33], [A14, A24, A34, A44]].*

A11	A12	A13	A14
A21	■	A23	A24
A31	A32	A33	A34
A41	A42	■	A44

A11	A21	A31	A41
A12	■	A32	A42
A13	A23	A33	■
A14	A24	A34	A44

FIGURA 5.1: Representación del tablero con dos matrices. A la izquierda la matriz por filas y a la derecha por columnas.

En las secciones siguientes se presentan los dos diseños desarrollados.

5.1. Diseño 1: Backtracking

Este modelo utiliza la técnica de *backtracking*, que permite recorrer ordenadamente el espacio de soluciones, junto con la técnica de *generación y chequeo*: un proceso genera soluciones candidatas y otro chequea que cumplan todas las condiciones. Las podas al espacio de soluciones se realizan lo antes posible, para evitar explorar soluciones parciales que nunca conducirán a una solución final (esto implica que el chequeo se realiza a medida que se va generando la solución).

Dividimos la generación de crucigramas en dos etapas secuenciales: la generación de la máscara y la asignación de palabras y definiciones a la misma (ver figura 5.2).

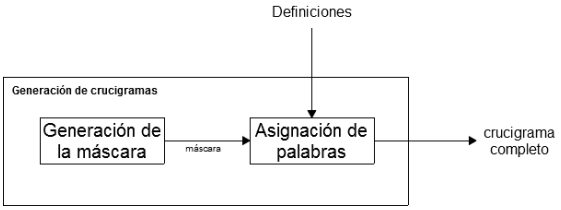

FIGURA 5.2: Diseño inicial del módulo de generación de crucigramas

La generación de la máscara implica establecer el tamaño de la cuadrícula y disponer algunos casilleros negros en la misma. Posteriormente, en la etapa de asignación de palabras, podríamos agregar más casilleros negros de ser necesario, por lo que decimos que aquí se genera una "máscara incompleta". Para ubicar los cuadrados negros tenemos en cuenta criterios de calidad, relativos a la cantidad de slots largos y cortos, y la cantidad de casilleros negros totales [2].

Inicialmente obtenemos una lista de todos los slots del tablero. Luego, a cada slot le asignamos una palabra. Esta asignación se hace alternando entre slots horizontales y verticales, para aumentar las posibilidades de que el tablero final sea válido. Si trabajáramos primero con todos los slots horizontales, por ejemplo, se podrían generar palabras inexistentes en los verticales, y esto se chequearía recién al empezar a trabajar con los slots verticales, por lo que habría que retroceder muchos pasos, lo que no es eficiente. El mecanismo de *backtracking* se encarga de ir probando una a una todas las combinaciones posibles hasta alcanzar una solución.

[2] Para más detalles referirse al apéndice C.

Problemas

Este algoritmo presenta el inconveniente de ser sumamente ineficiente para tableros de 7x7 o superiores. Esta ineficiencia, que redunda en tiempos de ejecución excesivamente altos, se debe a que:

1. la cantidad de palabras candidatas para cada slot es muy grande, por lo que hay demasiadas posibilidades para cada slot
2. la elección de la colocación de casilleros negros en un slot en caso de que no haya una palabra que se adecue a él, da lugar a muchas posibilidades nuevas para probar
3. se prueban muchísimas combinaciones ya probadas varias veces, cada vez que falla una palabra

Por ello, se diseñó otro algoritmo que emplea la técnica greedy, y se describe a continuación.

5.2. Diseño 2: Greedy

La estrategia *greedy* asegura la construcción de un crucigrama válido en cada paso del proceso de generación. De esta forma, el algoritmo no tarda indefinidamente en encontrar una solución y resolvemos el problema anterior.

Consideramos tres conjuntos independientes de palabras ordenados por prioridad:

1. Las palabras extraídas a partir de un definendum de una sola palabra.
2. Las palabras extraídas a partir de un definendum de más de una palabra.
3. Las palabras obtenidas del módulo de recursos externos.

En primer lugar, elegimos la primera palabra a colocar del primer conjunto, de largo cercano a N (dimensión del tablero), y su ubicación (determinada de forma aleatoria).

En cada paso del algoritmo, sorteamos el conjunto de palabras a considerar y la dirección (alternando entre horizontal o vertical). Los conjuntos 1 y 2 se eligen con probabilidad de 0,7 y 0,3 respectivamente[3], mientras que las palabras del tercer conjunto se utilizan cuando ya no es posible asignar palabras de los conjuntos anteriores.

[3]Estos valores fueron elegidos heurísticamente para aumentar la calidad de las pistas.

Luego, debemos elegir cuál de las palabras del conjunto colocaremos y su posición en el tablero, ya que eventualmente puede existir más de un lugar compatible. Estos valores se hallan maximizando un puntaje, dado por:

1. la cantidad de palabras en la dirección opuesta que intersecta la palabra elegida
2. la cantidad de palabras nuevas que se forman (todas las palabras resultantes deben ser válidas)

Cuando ya no podemos colocar palabras en el tablero, completamos los casilleros en blanco con casilleros negros. Podemos afirmar que el crucigrama generado es válido, ya que por construcción siempre se obtiene un crucigrama válido en cada paso.

Finalmente, obtenemos las pistas asociadas a las palabras colocadas. Tomamos la decisión de realizar esta búsqueda de definiciones una vez completado el crucigrama, para no tener que mantener información no relevante para la asignación durante el procedimiento.

A continuación, presentamos el pseudocódigo del algoritmo.

Algoritmo 2 generar crucigrama

Entrada: N
Salida: Tablero, Asignadas, Pistas

```
generar_tablero(N,Tablero),
obtener_palabras(Unipalabras,Multipalabras,REpalabras),
ubicar_palabra_inicial(Unipalabras,Tablero),
% Asignar el resto de las palabras en el tablero
% "Asignadas" son las palabras colocadas en el tablero
asignar_palabras(vertical,Tablero,Unipalabras,Multipalabras,REpalabras,Asignadas),
asignar_casilleros_negros_faltantes(Tablero)
obtener_pistas(Asignadas,Pistas)
```

Algoritmo 3 asignar palabras

Entrada: Direccion, Tablero, Unipalabras, Multipalabras, REpalabras, Asignadas, prob_unipalabra
Salida: Tablero

```
% Seleccionar el conjunto de palabras a considerar (Palabras): se elige el conjunto 1 con probabilidad de prob_unipalabra y el 2 con probabilidad de (1-prob_unipalabra). El conjunto 3 se elige cuando ya no se pueden asignar palabras de los otros dos
seleccionar_conjunto(Unipalabras,Multipalabras,REpalabras, Palabras)
% Hallar la palabra P y posición que maximiza el puntaje, si falla al encontrar palabras compatibles se prueba con los otros conjuntos en orden de prioridad
obtener_mejor_palabra(Tablero,Direccion,Palabras,P,Posición)
colocar_palabra(P,Posición,Tablero)
% Retirar la palabra asignada (P) de los conjuntos, si P pertenece al conjunto Multipalabras se retiran todas las palabras extraídas por el mismo definendum, para no repetir la definición en las pistas
% Agregar P a la lista de palabras asignadas
obtener_conjuntos_nuevos(P,Unipalabras, NUpalabras, Multipalabras, NMpalabras, REpalabras, NREpalabras, Asignadas, NAsignadas),
obtener_direccion_nueva(Tablero,Direccion,NDireccion),
% Asignar el resto de las palabras
asignar_palabras(NDireccion,Tablero,NUPalabras,NMpalabras,NREpalabras,NAsignadas)
```

Ilustraremos el algoritmo por medio de un ejemplo.

Consideremos los siguientes conjuntos de palabras:

1. {ZINC, IIDH, INIA, ARCO}
2. {CO}
3. {AYO, NY}

y el tablero:

A11	A12	A13	A14
A21	A22	A23	A24
A31	A32	A33	A34
A41	A42	A43	A44

FIGURA 5.3: Tablero 4x4 vacío

Asignamos la primera palabra al tablero, eligiendo aleatoriamente el slot y su ubicación dentro del mismo:

A11	A12	A13	A14
A21	A22	A23	A24
Z	**I**	**N**	**C**
A41	A42	A43	A44

FIGURA 5.4: Tablero 4x4: asignación de la palabra inicial

A continuación, debemos colocar una palabra verticalmente, y sale sorteado el conjunto 1. Calculamos entonces la siguiente tabla:

Palabra	**Dirección**	**Posicionescompatibles**	**Puntajes**	**Posición y puntaje máximo**
IIDH	Vertical	-	-	-
INIA		[A12,A22,I,A42]	1 (intersecta a ZINC)	[A12,A22,I,A42] - 1
ARCO		[A14,A24,C,A44]	1 (intersecta a ZINC)	[A14,A24,C,A44] - 1

El puntaje máximo es 1 y colocamos, por lo tanto, la palabra 'INIA' en la posición [A12,A22,I,A42].

Quedan sin asignar las palabras:

1. {IIDH, ARCO}
2. {CO}

3. {AYO, NY}

Ahora debemos elegir una palabra para colocar en posición horizontal. Supongamos que nuevamente sale sorteado el conjunto 1. Siguiendo el mismo procedimiento, colocamos 'IIDH' en el slot [A11,I,A13,A14], quedando así el siguiente tablero:

I	I	D	H
A21	N	A23	A24
Z	I	N	C
A41	A	A43	A44

FIGURA 5.5: Tablero 4x4: asignación de palabra 'IIDH'

En el siguiente paso sale sorteado el conjunto 2 y colocamos la palabra 'CO' en dirección vertical, en las celdas [C,A44] del slot [H,A24,C,A44]. Al no poder asignar más palabras extraídas, procedemos a colocar las palabras del tercer conjunto. De esta manera, colocamos 'AYO' en las celdas [A,A43,O] y la palabra 'NY' en el slot [D,A23,N,Y].

Al no poder colocar más palabras en el tablero, culmina la asignación. El tablero con las palabras asignadas queda:

I	I	D	H
A21	N		
Z	I	N	C
	A	Y	O

FIGURA 5.6: Tablero 4x4: fin de la asignación

Por último, rellenamos los casilleros sin asignar con casilleros negros:

I	I	D	H
	N		
Z	I	N	C
	A	Y	O

FIGURA 5.7: Crucigrama 4x4 completo

Capítulo 6

Detalles técnicos

En este capítulo se explican aspectos de la implementación y se describen las herramientas utilizadas.

6.1. Módulo de extracción de definiciones

El procesamiento de un texto implica varias etapas, como se muestra esquemáticamente en la figura 6.1

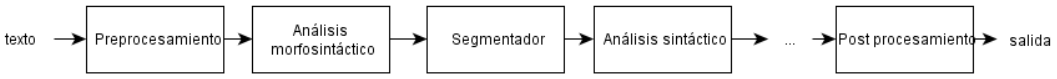

FIGURA 6.1: Etapas del PLN

A continuación se detallarán los principales puntos de su implementación.

Preprocesamiento

La fase de *preprocesamiento* fue corta ya que los textos empleados estaban limpios y se encontraban almacenados en el formato adecuado (texto plano). Fueron necesarios los siguientes ajustes:

1. Cambio de *encoding* de los archivos (de ANSI a utf8 sin BOM).
2. Agregado de puntos al final de las oraciones que carecían de signos de puntuación finales (sobre todo en títulos).

3. Cambio de paréntesis rectos por paréntesis curvos, para que no interfirieran con el reconocimiento de sintagmas.

4. Concatenación de archivos pequeños, de un mismo medio de prensa, para formar un único archivo, obteniendo así una mayor eficiencia en su procesamiento.

Se eligió trabajar en todas las capas y en todo momento con el mismo *encoding* para unificar el trabajo y evitar problemas en la conversión de formatos. Se eligió utf8 debido a que es el encoding por defecto para Python, nuestro lenguaje de programación principal, y a que se adecua a los caracteres propios de nuestro idioma.

Análisis morfosintáctico

Se eligió FreeLing ([37]) como herramienta para ejecutar todos estos pasos, sobre el texto íntegro. Al invocar al PoS tagger de FreeLing, se ejecuta una tokenización previa.

Luego de realizar algunas pruebas, vimos que era necesario realizar los siguientes ajustes en la configuración de FreeLing:

- *nombres propios*: FreeLing reconoce como un único token, correspondiente a un nombre propio, a las secuencias de palabras que inician por mayúscula y están eventualmente unidas por la preposición "de". Observando las definiciones extraídas, se notó que en la mayoría de las ocasiones también era necesario tener en cuenta las palabras iniciadas en mayúscula conectadas por la conjunción "y", como por ejemplo en el caso de *Instituto Brasileño de Geografía y Estadística*, por lo que se configuró para que así fuera.

- *locuciones*: se agregaron algunas locuciones a la lista de FreeLing, como ser "por su cuenta", "relativo a".

La salida del PoS tagger fue post-procesada borrando una columna con datos probabilísticos que no es compatible con el formato de entrada para ClaTex, ni es necesaria para nuestro proyecto.

Análisis sintáctico

Primero se hace un análisis sintáctico parcial con ClaTex y luego se usa el *full parser* de FreeLing, sobre las proposiciones más pequeñas que contienen el texto de interés. Esto podría provocar que se cometan errores en el parsing, ya que no se cuenta con la oración entera, ni mucho menos con contexto más allá de la oración, pero en la práctica

se observó que los resultados son mejores con dicha segmentación que sin ella, por lo que se hace así.

Se eligió el *full parser* ya que las otras opciones de parsing de FreeLing son inadecuadas: el *shallow parsing* aporta muy poca información de la estructura subyacente (devuelve un árbol "muy aplanado"), y el *dependency parsing* se implementa como una transformación de la salida del *full parsing* [1] con un formato distinto, que agrega información como por ejemplo la función sintáctica (sujeto, objeto directo, etc.), cuyo uso podría ser de utilidad pero está fuera del alcance de este proyecto.

Post procesamiento

Para la obtención de los nombres más frecuentes tomamos el texto taggeado y obtuvimos una lista con todos los nombres comunes (no propios), ya que consideramos a los nombres propios de interés aunque sean frecuentes. Luego, hallamos la frecuencia de cada uno en el conjunto de los corpus de desarrollo, utilizando *NLTK*, y los guardamos en una lista en orden decreciente, es decir, del más al menos frecuente. De manera experimental, definimos que un nombre es muy frecuente si se encuentra dentro de los primeros 50 de dicha lista.

6.2. Módulo de generación de crucigramas

Se utilizó Prolog como lenguaje de implementación para los dos diseños. Se eligió este lenguaje inicialmente porque provee el algoritmo de *backtracking* implícito, lo que facilita la implementación del diseño 1. A pesar de que dicho diseño se descartó, el lenguaje continuaba siendo adecuado debido a que brinda un mecanismo de unificación en orden 1, lo que aporta gran eficiencia para ambos diseños.

6.3. Persistencia

Para guardar los patrones y las definiciones encontradas se utilizó una base de datos MySQL, cuyo esquema se muestra en la figura 6.2.

[1] El *full parsing* provee un análisis de la estructura de la oración lo más detallado posible, mientras que el *shallow parsing* es menos detallado, usa distinciones menos finas en cuanto a los tipos de constituyentes, ignorando por ejemplo su estructura interna y su rol en la oración principal. El *dependency parsing* se basa en las relaciones de dependencia entre las palabras (como ser las funciones sintácticas), no existiendo el concepto de constituyentes.

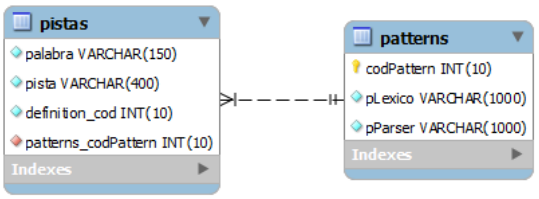

FIGURA 6.2: Esquema de la base de datos.

Se cuenta con una tabla *pistas* para los pares $\prec$ *pista* , *palabra* $\succ$ donde se almacenan esos dos datos, y además un identificador de la definición de la cual provienen.

Los patrones se almacenan en una tabla *patterns*, donde se persiste el código que identifica a cada patrón, y sus dos partes: *patrón léxico* y *patrón parser*.

Se configuró la base de datos para utilizar el *encoding* utf8 (*collation* utf8-unicode-ci) - el mismo que en el resto del proyecto -, tanto en sus tablas como en sus conexiones con la capa lógica.

En el caso de la persistencia de patrones fue necesario realizar varias transformaciones para llevar el objeto *Pattern* a una base de datos relacional sin pérdida de información y de manera unívoca. Además se tuvo en cuenta que se mantuviera la integridad de las expresiones regulares embebidas en el *patrón parser*.

6.4. Evaluación de herramientas

En esta sección se exponen las herramientas utilizadas durante el desarrollo de este proyecto (6.1), describiendo ventajas y desventajas de las principales, y los problemas que éstas originaron.

Python

Ventajas: Ofrece muchas facilidades para el manejo de strings y expresiones regulares, es un lenguaje "compacto", del cual ya se poseía cierto conocimiento previo, y ofrece librerías para trabajar con otras herramientas, como NLTK.

Desventajas: Durante el desarrollo de este proyecto no se le encontraron desventajas.

Función	Nombre de la herramienta
Lenguaje de programación	Python v2.7 para el módulo extracción de definiciones y comunicación con WordNet. SWI Prolog v6.6.6 para la generación del crucigramas.
Análisis morfosintáctico. Reconocimiento de entidades con nombre.	FreeLing v3.1 español (PoS tagger, full parsing, named entity classification).
Separador en proposiciones	Clatex v2006
Tesauro	WordNet español v3.0
Manejador de base de datos	MySQL v5.6
Paquete de expresiones regulares recursivas	Python regex v2014.12.24
Contar frecuencias de palabras en corpus, extracción de información de tesauro.	NLTK v3.0.2

TABLA 6.1: Tabla de herramientas

Clatex

Ventajas: Ofrece buena integración con Python y buenos resultados en general, además de contar con soporte técnico directo, ya que fue desarrollado por el Grupo PLN de la Facultad de Ingeniería de la UdelaR.

Desventajas: Devuelve resultados erróneos en algunas ocasiones, como ser desbalanceo de paréntesis o comillas. Además, debido a algunos detalles en su implementación, es compatible solo con *Windows*.

FreeLing

Ventajas: Es considerado una de las mejores herramientas disponibles para el español. Ofrece una suite variada de herramientas para el procesamiento del lenguaje natural, evitando así los problemas de compatibilizar el formato si se utilizaran varias herramientas distintas.

Desventajas: Desde el punto de vista de comunicaciones, no se encontró una manera sencilla para invocarlo como librería desde Python (utilizando *Windows*), lo que redundó en problemas de eficiencia.

Full parser de FreeLing

El parser presenta varios errores, incluso en oraciones pequeñas, a saber:

- no parsea correctamente cuando el texto a analizar tiene signos de puntuación desbalanceados (por ejemplo faltan las comillas que cierran)
- tiene muchos errores debido al problema de PP attachment, como se ilustra en el ejemplo 6.1
- no parsea correctamente las conjunciones, especialmente si se trata de un listado de 3 o más elementos
- a veces desordena palabras (cambia su posición dentro de la oración)

Ejemplo 6.1. *Ejemplos de parsing incorrectos*
El texto

"El oportuno aviso de un conductor que observó a la mujer sobre el paso elevado y la intervención de bomberos evitaron el accidente"

tiene un problema de ambigüedad del grupo preposicional "sobre el paso elevado" (PP attachment problem). Podría interpretarse que el conductor observó desde el paso elevado a la mujer ("sobre el paso elevado" modifica a "observó"), o que la mujer estaba sobre el paso elevado, y el conductor la observó ("sobre el paso elevado" modifica a "la mujer"). Esta última interpretación parece ser la más verosímil para este caso. Esta clase de ambigüedades causan comúnmente análisis poco adecuados en los parsers. Por ejemplo, FreeLing interpretó erróneamente dicho fragmento, como se ve en la figura 6.3.

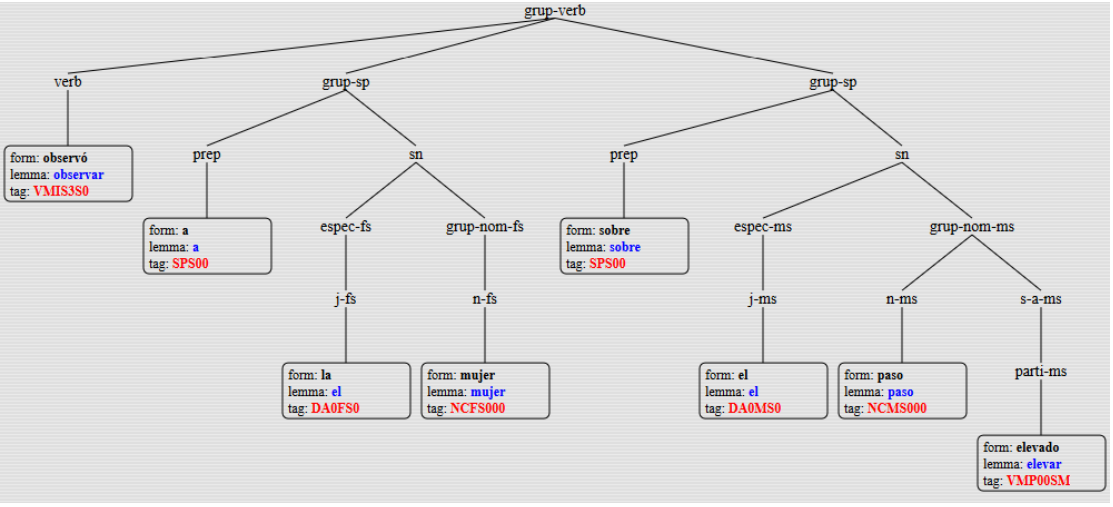

FIGURA 6.3: Ejemplo de parsing incorrecto

El análisis sintáctico de "Gregory participó de la guerrilla en los años sesenta" devuelve el árbol de la figura 6.4. Obsérvese que la palabra "sesenta" debería pertenecer al mismo sintagma nominal que "los años".

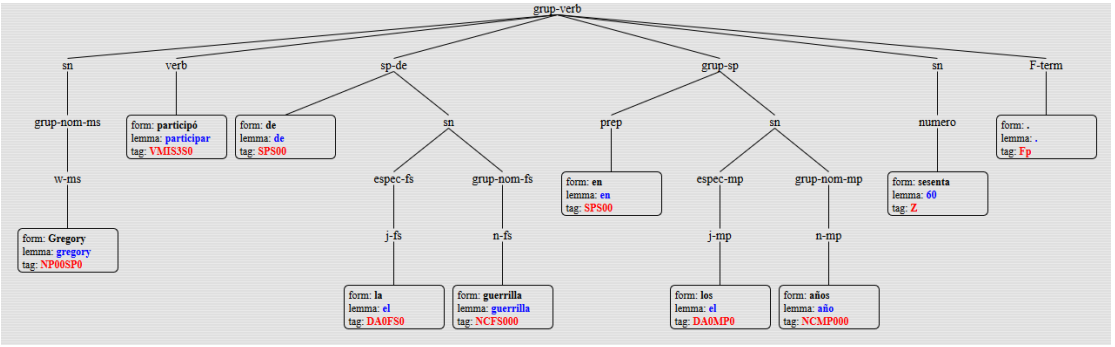

FIGURA 6.4: Ejemplo de parsing incorrecto

El análisis sintáctico de "Referirse a otros impactos nocivos, por ejemplo la erosión del suelo causada por la siembra intensiva de productos como la soja, o el empleo de agroquímicos, que contaminan el agua", cambia de lugar el fragmento "de agroquímicos,", dejando el texto resultante como "[...] o el empleo que contaminan el agua de agroquímicos,", como se aprecia en la figura 6.5

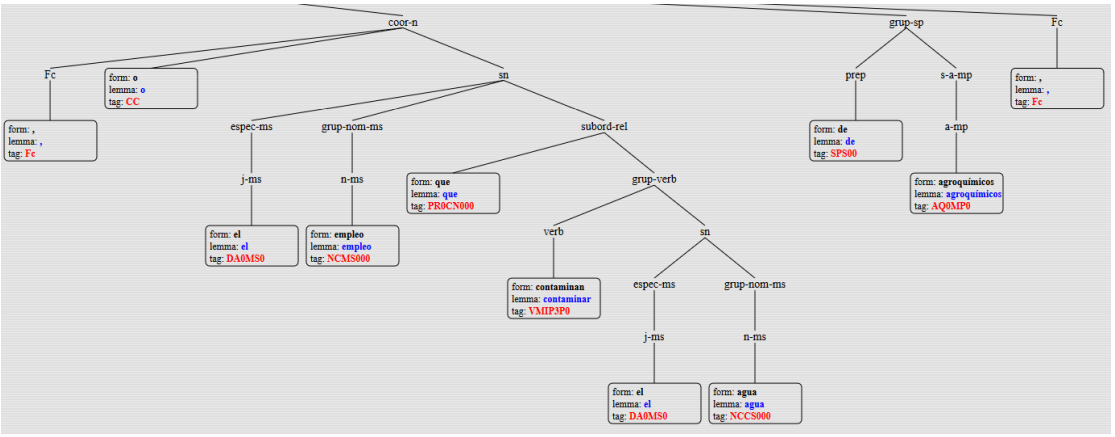

FIGURA 6.5: Ejemplo de parsing incorrecto

Resolución de correferencias de FreeLing

FreeLing ofrece una herramienta para resolver correferencias [2] , lo que podría ser muy útil para encontrar más definiciones (dentro de una misma oración y también definiciones a partir de oraciones contiguas), pero lamentablemente sus resultados son muy malos, por lo que hubo que descartar esta idea.

[2] La *resolución de correferencias* (o relaciones anafóricas) refiere a encontrar a qué entidad, aparecida anteriormente en el texto, se está haciendo referencia con una palabra o expresión en particular. Por ejemplo al decir "Ayer vi una casa. Su cocina era enorme", vemos que "su" señala implícitamente que la cocina es la de la casa previamente nombrada.

Prolog

Ventajas: Ofrece backtracking y unificación automáticamente y de gran eficiencia, lo que lo hace muy adecuado para el problema tratado. Además es un lenguaje sencillo del cual ya se tenía un buen manejo.

Desventajas: Es lento si el espacio de soluciones es grande.

WordNet español

Ventajas: Tiene buena integración con NLTK y Python, lo que facilita la extracción de sus definiciones. Ya se contaba con conocimiento previo de esta herramienta. Ofrece datos sobre las relaciones entre las palabras, por lo que podría ser usado para trabajos futuros.

Desventajas: En su versión en español, tiene muchos lemas sin definición, por lo que no fueron de utilidad para este proyecto.

Capítulo 7

Evaluación y resultados

7.1. Extracción de definiciones

7.1.1. Pruebas realizadas

Los corpus utilizados fueron:

Corpus	**Tamaño** (palabras)	**Observación**
Historia	19457	Conjunto de textos históricos
180	7500	Conjunto de textos extraídos del portal de noticias 180
Montevideo Portal	33160	Conjunto de textos extraídos del portal de noticias Montevideo Portal
Corín	101940	Conjunto de textos periodísticos uruguayos del período 1996-2000
Observador	20024	Conjunto de textos extraídos del diario El Observador
Espectador	233505	Conjunto de textos extraídos del diario El Espectador
República	730132	Conjunto de textos extraídos del diario La República
Francia	21271	Conjunto de textos extraídos manualmente a partir de distintos diarios nacionales e internacionales (en español), basados en noticias del atentado contra Charlie Hebdo en Francia el 7 de enero del 2015
Held Out	20128	Conjunto de textos extraídos manualmente de los portales de noticias 180 y Montevideo Portal
Testing	23581	Conjunto de textos extraídos manualmente de los diarios El País y La República

Tabla 7.1: Corpus utilizados

Las pruebas fueron realizadas por etapas, mejorando de forma iterativa el sistema. A continuación, desarrollaremos a grandes rasgos las pruebas y los resultados de cada etapa.

Primera etapa

En una primera instancia generamos manualmente 14 patrones basándonos en los textos a analizar. Teniendo en cuenta solamente estos patrones el sistema logró extraer un conjunto restringido de definiciones, como se puede apreciar en la siguiente tabla:

Corpus	Cantidad definiciones
Historia	8
180	10
Montevideo Portal	22
Corín	42
Observador	31

TABLA 7.2: Primera etapa: cantidad de definiciones por corpus

Cabe mencionar que la mayoría de estas definiciones provenía fundamentalmente de patrones basados en paréntesis o comas.

Ejemplo

1. "*El Ministerio de Salud Pública (MSP)...*"
2. "*El presidente de la República, José Mujica...*"

A su vez, no se encontraron nuevos patrones interesantes, problema que se le adjudicó a la pobre extracción y a la naturaleza de los textos utilizados.
Los patrones originales no fueron capaces de extraer gran cantidad de definiciones debido a que eran demasiado específicos, por lo que surgió la necesidad de generalizarlos.

Segunda etapa

En este segundo conjunto de pruebas partimos de 10 patrones, resultado de la generalización de los mismos en la etapa anterior.

A pesar de que el volumen de extracción se duplicó, el resultado no fue satisfactorio. Aumentó considerablemente la cantidad de falsos positivos tanto en las definiciones extraídas como en los patrones encontrados.

Debido a que las definiciones provenían de patrones muy similares y los textos presentaban gran cantidad de definiciones con diferente estructura, aumentamos la cantidad de patrones iniciales. Al realizar esto, se incrementó de forma considerable el volumen de definiciones extraídas.

Corpus	Cantidad definiciones
Corín	352
180	31
Espectador	404
Montevideo Portal	150
Historia	54
Observador	125

TABLA 7.3: Segunda etapa: cantidad de definiciones por corpus

Observamos que el sistema presentaba un rendimiento muy bajo. En una siguiente iteración optimizamos las expresiones regulares utilizadas y generamos más patrones iniciales. Como resultado, se obtuvo un total de 322 definiciones para el corpus 180 (el más chico). Notamos que la cantidad de falsos positivos se incrementó sustancialmente y el sistema no fue capaz de encontrar patrones nuevos. Debido a esto, tuvimos que modificar y descartar algunos patrones que generaban mucho ruido, pero los resultados no fueron satisfactorios.

Tercera etapa

Para mejorar los resultados de la etapa anterior, realizamos un análisis de la calidad de cada patrón, descartando aquellos que no extraían buenas definiciones y especificando los que generaban gran cantidad de falsos positivos.

En esta etapa, trabajamos fundamentalmente con los corpus *180* y *Montevideo Portal*.

Para el análisis, utilizamos 32 patrones iniciales y medimos la precisión total y por patrón. Además, anotamos manualmente el corpus *180*, buscando las definiciones para poder medir el recall del sistema. El análisis del corpus fue realizado por dos anotadores por separado, llegando luego a un acuerdo (*Inter annotator agreement*) del 96 %.

En una primera ejecución con el corpus *180*, se obtuvieron 174 definiciones, 0.45 de precisión y 0.67 de recall. En la gráfica 7.1 podemos apreciar la precisión por patrón (teniendo en cuenta los patrones que extrajeron definiciones).

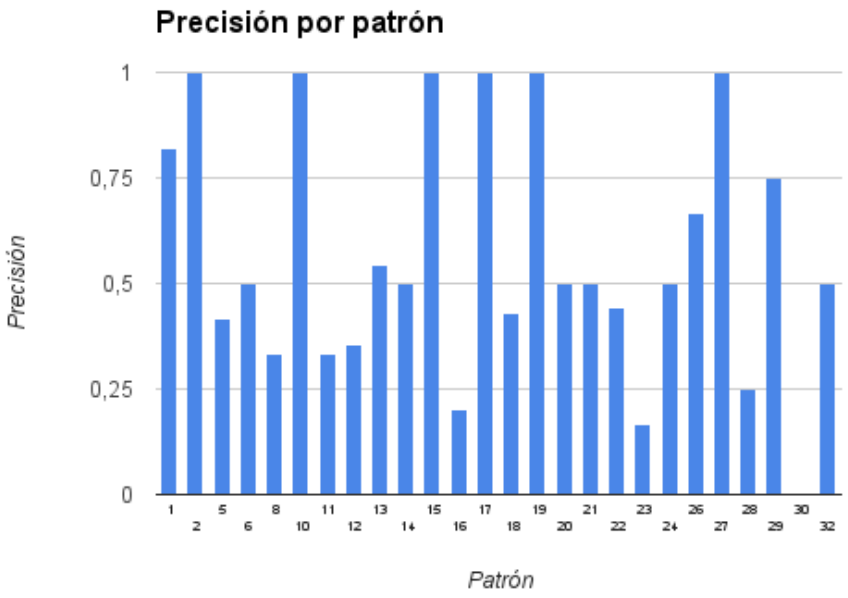

FIGURA 7.1: Corpus 180: Precisión por patrón inicial

Notamos que hay varios patrones de baja precisión, los cuales analizamos generando patrones nuevos a partir de éstos o descartándolos, según fuera conveniente. Una vez hecho esto, realizamos una segunda ejecución y medimos nuevamente.

En la gráfica 7.2 se puede ver la mejora en la calidad de los patrones.

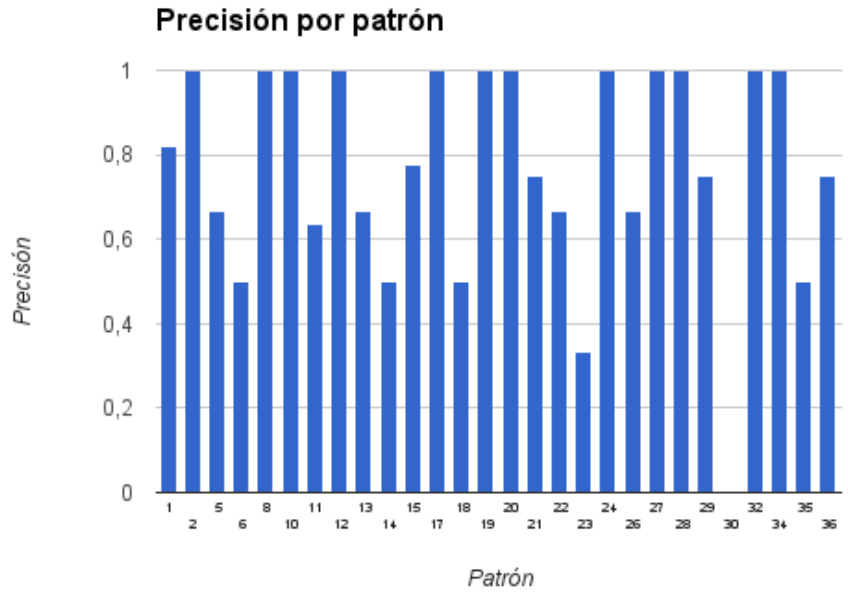

FIGURA 7.2: Corpus 180: Precisión por patrón mejorada

Para esta segunda instancia, obtuvimos 114 definiciones, una precisión de 0.73 y un recall de 0.70.

Corpus 180			
True Positives:	83	False Negatives:	36
False Positives:	31	True Negatives:	-

Tabla 7.4: Tercera etapa: resultado para el corpus *180*

El corpus *Montevideo Portal* arrojó otros resultados, extrayendo 474 definiciones y 0.57 de precisión. A partir del análisis de estos resultados, removimos los patrones con baja precisión (número 16, 25 y 30) de los patrones de base.

Cuarta etapa

En esta etapa, se tuvieron en cuenta los corpus *180* y *Montevideo Portal*. Al notar que muchas definiciones no eran adecuadas para formar parte de un crucigrama, decidimos realizar un post procesamiento de las mismas (como se explicó anteriormente en la sección 4.1.4). De esta manera, le dimos un formato adecuado a aquellas extracciones que constituían una definición y descartamos gran cantidad de definiciones que no eran adecuadas.

El resultado de esta etapa fue mucho mejor que en las anteriores, logrando para el corpus *180* una precisión de 0.80 y para *Montevideo Portal* 0.70.

Corpus 180		Precisión
True Positives	81	
False Positives	20	0,80
TOTAL	101	

Tabla 7.5: Cuarta etapa: resultado para el corpus *180*

Corpus Montevideo Portal		Precisión
True Positives	267	
False Positives	116	0,70
TOTAL	383	

Tabla 7.6: Cuarta etapa: resultado para el corpus *Montevideo Portal*

A pesar de las mejoras en los resultados, aún no se logró encontrar nuevos patrones de relevancia mediante bootstrapping. La explicación que se le encontró a este fenómeno fue que la naturaleza propia de los textos no era la adecuada, ya que una vez definido un término en el texto, no se volvía a definirlo (cada corpus pertenece a un solo texto de prensa). Para un mejor análisis de este problema, creamos manualmente el corpus *Francia* (de 21271 palabras) a partir de artículos de diferentes diarios, basados en noticias del atentado contra *Charlie Hebdo* en Francia. Al estar basado en la misma noticia de diferentes diarios, es coherente asumir que términos similares serán definidos en los distintos textos y, por lo tanto, la probabilidad de encontrar nuevos patrones aumenta. Sin embargo, el resultado no fue exitoso. Cabe mencionar que utilizando textos que contienen más de una forma de definir los mismos términos, el sistema sí es capaz de extraer dichos patrones.

Considerando los fragmentos de texto "*El hombre es un animal racional.*" y "*El hombre es conocido como un animal racional.*", el sistema es capaz de extraer el patrón "*es conocido como*" a partir del patrón "*es*".

Por último, se siguen presentando los falsos positivos. Muchos de ellos se deben a errores del full parser de FreeLing, considerando una estructura que el texto no posee realmente y extrayendo así definiciones erróneas.

Quinta etapa

Para la ejecución de estas pruebas, creamos el corpus Held Out (20218 palabras) recuperando manualmente artículos de los portales de noticias *180* y *Montevideo Portal*.

En la siguiente tabla se muestran los resultados obtenidos:

Held Out Corpus		Precisión
True Positives	142	
False Positives	63	0,69
TOTAL	205	

TABLA 7.7: Quinta etapa: resultado inicial para el corpus *Held out*

Se obtuvo un total de 205 pares $\prec$ *definición* , *definendum* $\succ$, los cuales generan 338 pares $\prec$ *pista* , *palabra* $\succ$ para ser utilizados por el *módulo de generación de crucigramas*.

Notamos la aparición de varios falsos positivos provenientes de un mismo patrón, por lo que ejecutamos nuevamente sacando el mismo de los patrones de base. El resultado de dicha ejecución fue el siguiente:

Held Out Corpus		Precisión
True Positives	129	
False Positives	43	0,75
TOTAL	172	

TABLA 7.8: Quinta etapa: resultado final para el corpus *Held out*

Se obtuvo un total de 172 pares $\prec$ *definición* , *definendum* $\succ$, los cuales generan 289 pares $\prec$ *pista* , *palabra* $\succ$ para ser utilizados por el *módulo de generación de crucigramas.*

Cabe mencionar que remover dicho patrón no afecta significativamente los resultados presentados en la etapa anterior.

Sexta etapa

Para este último conjunto de pruebas, creamos un corpus de testeo (23518 palabras) recuperando manualmente artículos de los diarios *El País* y *La República*. El resultado de la ejecución se muestra en la siguiente tabla:

Corpus de testeo		Precisión
True Positives	159	
False Positives	59	0,73
TOTAL	218	

TABLA 7.9: Sexta etapa: resultado para el corpus *Testing*

El sistema extrajo un total de 218 pares $\prec$ *definición* , *definendum* $\succ$, los cuales generan 410 pares $\prec$ *pista* , *palabra* $\succ$ para ser utilizados por el *módulo de generación de crucigramas.*

Se puede apreciar que el resultado no difiere significativamente del obtenido en la etapa anterior.

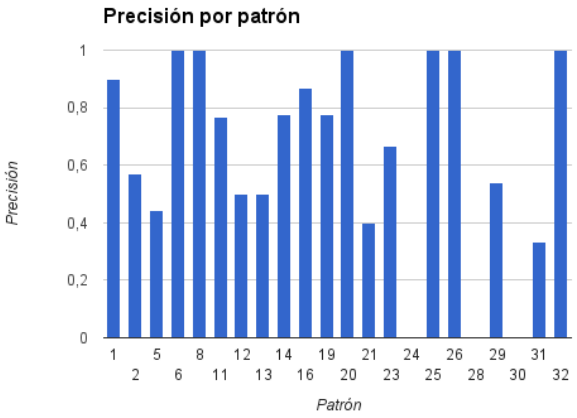

FIGURA 7.3: Corpus de testeo: Precisión por patrón

En la gráfica 7.3 vemos que los patrones con peor precisión son el 24, 28 y 30.

Es importante destacar que, como se muestra en la gráfica 7.4, la cantidad de definiciones extraídas por estos patrones es muy pequeña (entre una y dos definiciones), por lo que no afecta de forma considerable el resultado final.

La gráfica 7.4 muestra la cantidad de definiciones extraídas por patrón.

FIGURA 7.4: Corpus de testeo: definiciones por patrón

Se puede apreciar en la gráfica 7.4 que los patrones 1 y 11 fueron los que extrajeron más definiciones, con una precisión de 0,9 y 0,77 respectivamente. En la tabla 7.10 se muestran esquemáticamente dichos patrones:

Número	Patrón léxico	Patrón parser	Ejemplo
1	[⟨_, _, {[Nombre], TRUE,1}⟩, ⟨_,_, {[Nombre], FALSE,5}⟩, ⟨(, (, {[Paréntesis], TRUE,1}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩, ⟨),), {[Paréntesis], TRUE,1}⟩]	[⟨≺DEF≻, _, sn⟩, ⟨(, (, Paréntesis⟩, ⟨≺DFM≻,[Nombre propio],grup-nom⟩, ⟨),), Paréntesis⟩]	*Ministerio de Salud Pública (MSP)* ...
11	[⟨_, _, {[Nombre Propio], TRUE,1}⟩, ⟨_, _, {[Nombre], FALSE,5}⟩, ⟨_, _, {[Verbo], TRUE,1}⟩, ⟨_, _, {[Nombre], FALSE,5}⟩, ⟨_, _, {[Nombre], TRUE,1}⟩]	[⟨≺DFM≻,[Nombre Propio], sn⟩, ⟨≺DFMC≻,_, grup⟩, ⟨_,_,Verbo⟩, ⟨≺DEF≻, _, sn⟩, ⟨≺DEFC≻,_, grup⟩]	*Mujica respaldó importante inversión minera.*

TABLA 7.10: Patrones con más definiciones

En esta última etapa tampoco se logró encontrar nuevos patrones mediante la técnica de *bootstrapping*, pero la mejora sucesiva de los patrones de base permitió obtener buenos resultados.

7.1.2. Resultados obtenidos

A continuación, se muestran algunos ejemplos de definiciones extraídas por el módulo de extracción.

En la tabla 7.11 se muestran casos de extracción exitosos, mientras que en la tabla 7.12 se muestran definiciones que contienen errores menores.

Palabra	Pista
IBGE	*Instituto Brasileño de Geografía y Estadística*
Alicia	*(_____ Martínez). La presidenta de la Sociedad Uruguaya de Glaucoma*
Judith	*(_____ Uturbey). La subdirectora del Hospital de Ojos*
Puntigliano	*Ex presidente de la Administración Nacional de Puertos*
Mujica	*Quien respaldó importante inversión minera*
Vaticano	*(El _____). Organización que defiende el celibato pese a críticas por casos de pedofilia*
Uruguay	*Lugar que obtendrá 80 millones del Focem a través de un préstamo*
Felinos	*(Los grandes _____). Especies carismáticas*
Soja	*(Maíz , _____ , café y caña de azúcar). Cuatro productos destinados sobre todo al mercado externo*
Glaucoma	*Segunda causa de ceguera en el mundo*
Daisy	*(_____ Tourné). La flamante ministra del Interior*
Auto	*Un carro*
Tokio	*Segunda plaza financiera del planeta*

TABLA 7.11: Definiciones extraídas con éxito

Palabra	Pista	Error
Canelones	*Organización a la que pertenece Marcos Carámbula*	Canelones es un lugar, no una organización (clasificación equivocada dada por Freeling)
CNT	*Organización relacionada con los dirigentes de la Convención Nacional de Trabajadores*	CNT es la Convención Nacional de Trabajadores, se identificó de forma errada el constituyente del definendum
Mercosur	*(Fondo de Convergencia Estructural del _____). Focem*	Intercambio de definición y definendum

Tabla 7.12: Definiciones extraídas con errores

Por otro lado, en la tabla 7.13 se pueden ver ejemplos de definiciones extraídas erróneamente.

Palabra	Pista	Error
India	*Es el segundo país*	Problemas con el parser de Freeling
Europa	*(En _____). Lugar en el que existen experiencias que utilizan métodos similares*	Se necesita del contexto para entender la definición
Libia	*Los países vecinos*	Se necesita del contexto para entender la definición
Honduras	*Laos y Camboya*	Problemas con el parser de Freeling

Tabla 7.13: Definiciones erróneas

7.2. Generación de crucigramas

7.2.1. Pruebas realizadas

Las primeras pruebas realizadas fueron sobre el módulo de generación de crucigramas implementado utilizando la técnica de *backtracking*, como se detalla en la sección 5.1.

Se ejecutó el algoritmo para las dimensiones 5x5, 6x6, 7x7 y 10x10. Los resultados de estas pruebas se pueden ver en la tabla 7.14.

Dimensión	Resultado
5x5	Se generó un crucigrama exitosamente. El mismo se muestra en la figura 7.5.
6x6	La ejecución finaliza sin encontrar crucigramas válidos.
7x7	La ejecución no finaliza en un tiempo razonable (se deben probar muchas combinaciones de palabras y su posición)
10x10	La ejecución no finaliza en un tiempo razonable

TABLA 7.14: Resultados iniciales: generación de crucigramas

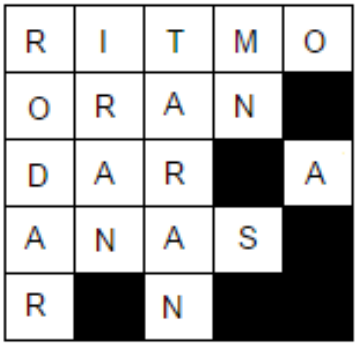

FIGURA 7.5: Crucigrama 5x5 generado con backtracking

Luego, considerando los problemas con la estrategia anterior, se cambió el módulo de generación de crucigramas utilizando un enfoque *greedy* y se realizaron las pruebas nuevamente. En esta ocasión, se generaron crucigramas válidos en todos los casos, pero se notó que las pistas presentaban en su mayoría el mismo formato ("(*José* _____). *Presidente de la República*"), afectando la calidad de las mismas. Por tanto, se dividieron las palabras y su pista correspondiente en categorías, como se indica en la sección 5.2. Una vez hecho esto, se generaron nuevamente los crucigramas, dándole prioridad a la asignación de palabras de determinada categoría. Los resultados de estas últimas pruebas se pueden ver en la siguiente sección.

7.2.2. Resultados obtenidos

A continuación, se muestran algunos de los crucigramas obtenidos por el *módulo de generación de crucigramas*, considerando 879 definiciones extraídas y 14497 definiciones de recursos externos.

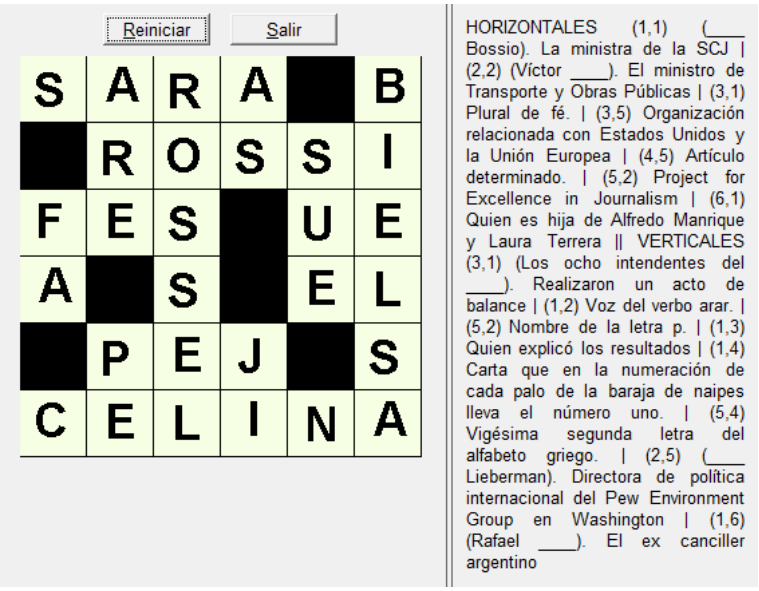

FIGURA 7.6: Crucigrama 6x6 generado con greedy

En la figura 7.6 se puede apreciar un crucigrama de dimensión 6x6, formado por un total de 15 palabras, 9 provenientes del *módulo de extracción de definiciones* y 6 de *recursos externos*.

FIGURA 7.7: Crucigrama 9x9 generado con greedy

La figura 7.7 muestra un crucigrama de dimensión 9x9, formado por 26 palabras, 12 extraídas por el *módulo de extracción de definiciones* y el resto de *recursos externos*.

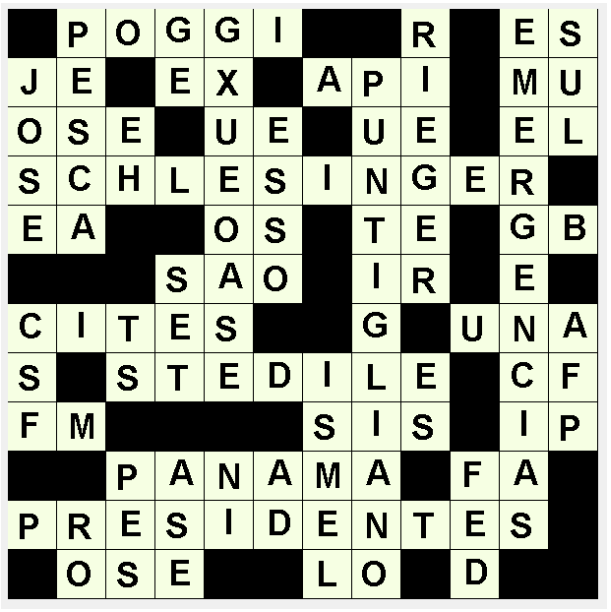

FIGURA 7.8: Crucigrama 12x12 generado con greedy

Por último, en la figura 7.8 se puede ver un crucigrama de dimensión 12x12, formado por 48 palabras, 22 provenientes del *módulo de extracción* y 26 de *recursos externos*.

Capítulo 8

Conclusiones y Trabajo Futuro

Resultados alcanzados y dificultades encontradas

Se logró construir un sistema de extracción de definiciones a partir de textos de prensa en español, el cual presenta una precisión de 73 %. Comparando los resultados obtenidos con los de la literatura actual, este resultado parece muy prometedor. En trabajos similares para otros idiomas, donde las reglas se construyen de forma manual a través de la observación, se alcanza un promedio de 46 % de precisión. Cuando solamente se usan las reglas más efectivas, la precisión aumenta a más del 70 %, por lo cual parecería que nos dirigimos por un buen camino. En el diseño inicial se planteó desarrollar este sistema utilizando *bootstrapping* para encontrar nuevos patrones y, por ende, más definiciones. En la práctica, no se logró que el *bootstrapping* descubriera nuevos patrones, lo cual traza un posible lineamiento de trabajo a futuro, pero la mejora de los patrones manuales permitió paliar esta situación.

El volumen de definiciones extraídas, si bien lo consideramos bueno, se puede incrementar mejorando los patrones de base, así como la naturaleza de las mismas. Las pruebas realizadas arrojaron resultados prometedores para el área de la extracción de información, pudiendo mejorar varios aspectos del sistema, simplemente modificando los patrones de base. No obstante, otras estrategias se deben explorar para mejorar el rendimiento.

A su vez, se construyó un sistema de generación de crucigramas a partir de las definiciones extraídas. Las pruebas realizadas a este sistema fueron exitosas, logrando generar en tiempo razonable un crucigrama completo. La calidad de los crucigramas construidos es buena, siendo aproximadamente un 50 % de las palabras utilizadas provenientes del *módulo de extracción de definiciones* y el resto de *recursos externos*. Inicialmente, se planteó la utilización de la técnica de *backtracking* para implementar la generación del

crucigrama, lo que mostró ser muy ineficiente para tableros medianos o grandes, por lo que se diseñó el algoritmo *greedy* que logra buenos resultados en tiempos adecuados.

La falta de herramientas precisas para el idioma español, constituyó el problema principal durante el desarrollo del sistema de extracción de definiciones. Si bien FreeLing resultó beneficioso para el análisis de los textos, demostró tener dificultades para realizarlo de forma precisa, obligándonos a tomar medidas para prevenir la propagación de los errores generados por la herramienta. Aún así, no fue posible eliminar completamente el impacto provocado por estas fallas, desmejorando la precisión del sistema construido.

Trabajo futuro

Consideramos que es imprescindible realizar análisis más detallados de los corpus, contando con más anotaciones manuales, posiblemente trabajando en conjunto con lingüistas, para desarrollar patrones más precisos y mejorar los niveles de recall. Esto podría permitir clasificar las definiciones en grupos y construir patrones especializados para cada uno. Se podría utilizar el sistema de parsing de dependencias para explotar la información adicional que este brinda; por ejemplo, el uso de las funciones sintácticas permitiría construir patrones más específicos. En esta línea, el utilizar una lista de lugares (con países y ciudades del mundo, lugares conocidos, etc.) podría ser de ayuda para mejorar el postprocesamiento de las definiciones, generando así pistas de crucigrama más amigables. En caso de seguir utilizando un módulo de recursos externos que toma información de un tesauro, se podrían explotar las relaciones entre las palabras para escribir las pistas.

Otro punto a mejorar es reducir el tiempo de ejecución. Para ello se podría explorar el usar FreeLing como librería integrada, y no como un comando externo, lo que implica un overhead en la invocación.

La utilización de reglas contextuales como línea para encontrar las definiciones parecería ser un área interesante para profundizar, así como también la utilización de herramientas como NLTK para manejar las proposiciones anidadas, y para manejar el árbol de parsing, en vez de utilizar las expresiones regulares. A su vez, se podría investigar el uso combinado con otras técnicas de machine learning, o incluso cambiar de paradigma y utilizar solo estas técnicas, y no las de reglas, para desarrollar el sistema de extracción de definiciones.

Por otra parte, una clasificación de las definiciones por temas según su semántica permitiría desarrollar un sistema de crucigramas temáticos.

Para aumentar el recall del sistema y ampliar el espectro de definiciones que se pueden encontrar, la construcción de una herramienta de resolución de correferencias parecería ser un elemento clave.

Finalmente, creemos que sería importante ampliar este trabajo para que los crucigramas puedan tener una aplicación educativa.

Referencias

Libros y artículos

[1] Aoife Aherne y Carl Vogel. *Crossing wordnet with crosswords, netting enhanced automatic crossword generation.* Inf. téc. TCD-CS-2005-52, 2005.

[2] Aoife Aherne y Carl Vogel. "Wordnet enhanced automatic crossword generation". En: *Proceedings of the 3rd International Wordnet Conference.* 2003, págs. 139-145.

[3] James Allen. "Estimation, Evaluations and Hold Outs". En: *Speech Recognition and Statistical Language Models Lectures.* 2003.

[4] Nguyen Bach y Sameer Badaskar. "A review of relation extraction". En: *Literature review for Language and Statistics II* (2007).

[5] Zafer Barutçuoğlu. "Automated Generation of Unconstrained Crossword Puzzles and an Estimate of Their Solution Space". En: *International Symposium on Computer and Information Sciences (ISCIS).* 1997.

[6] Eduardo Blanco, Nuria Castell y Dan Moldovan. "Causal Relation Extraction". En: *LREC.* 2008.

[7] Claudia Borg, Mike Rosner y Gordon Pace. "Evolutionary algorithms for definition extraction". En: *Proceedings of the 1st Workshop on Definition Extraction.* Association for Computational Linguistics. 2009, págs. 26-32.

[8] Serrana Caviglia et al. "Un sistema para la segmentación en proposiciones de textos en español". En: *Letras de hoje* 41.144 (2006).

[9] Yee Seng Chan y Dan Roth. "Exploiting syntactico-semantic structures for relation extraction". En: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1.* Association for Computational Linguistics. 2011, págs. 551-560.

[10] H. Cunningham. "Encyclopedia of Language & Linguistics". En: ed. por Keith Brown. Oxford: Elsevier, 2006. Cap. Information Extraction, Automatic.

[11] Lukasz Degórski, Michal Marcińczuk y Adam Przepiórkowski. "Definition Extraction Using a Sequential Combination of Baseline Grammars and Machine Learning Classifiers". En: *LREC*. Citeseer. 2008.

[12] Rosa Del Gaudio y António Branco. "Automatic extraction of definitions in portuguese: A rule-based approach". En: *Progress in artificial intelligence*. Springer, 2007, págs. 659-670.

[13] Jakob Engel et al. "On computer integrated rationalized crossword puzzle manufacturing". En: *Fun with Algorithms*. Springer. 2012, págs. 131-141.

[14] Anthony Fader, Stephen Soderland y Oren Etzioni. "Identifying relations for open information extraction". En: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011, págs. 1535-1545.

[15] Mariela Grassi et al. "Corpus informatizado: textos del español del Uruguay (CORIN)". En: *SLPLT-2-Second International Workshop on Spanish Language Processing and Language Technologies-Jaén, España*. 2001.

[16] Sanda Harabagiu, Cosmin Adrian Bejan y Paul Morărescu. "Shallow semantics for relation extraction". En: *Proceedings of the 19th international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc. 2005, págs. 1061-1066.

[17] Marti A Hearst. "Automatic acquisition of hyponyms from large text corpora". En: *Proceedings of the 14th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics. 1992, págs. 539-545.

[18] Daniel Jurafsky y James Martin. *Speech and language processing an introduction to natural language processing, computational linguistics, and speech*. 2nd. Prentice Hall, 2008.

[19] Margarita Lozano Pérez. "Extracción supervisada de relaciones entre conceptos a partir de texto libre en español o en inglés". En: *Universidad Rey Juan Carlos* (2012).

[20] Christopher D Manning, Prabhakar Raghavan e Hinrich Schütze. *Introduction to information retrieval*. Vol. 1. Cambridge university press Cambridge, 2008.

[21] Rosa María Ortega Mendoza. "Descubrimiento Automático de Hipónimos a partir de Texto no Estructurado". Tesis de lic. México: Instituto Nacional de Astrofísica, Óptica y Electrónica, 2007.

[22] Alexander Panchenko, Sergey Adeykin y Alexey Romanov. "Extraction of semantic relations between concepts with knn algorithms on wikipedia". En: *Concept Discovery in Unstructured Data Workshop (CDUD) of International Conference On Formal Concept Analysis, Belgium*. Citeseer. 2012, págs. 78-88.

[23] Adam Przepiórkowski et al. "Towards the automatic extraction of definitions in Slavic". En: *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing: Information Extraction and Enabling Technologies*. Association for Computational Linguistics. 2007, págs. 43-50.

[24] Bali Ranaivo-Malançon et al. "Automatic generation of fill-in clues and answers from raw texts for crosswords". En: *Information Technology in Asia (CITA), 2013 8th International Conference on*. IEEE. 2013, págs. 1-5.

[25] Leonardo Rigutini et al. "A Fully Automatic Crossword Generator". En: *Machine Learning and Applications, 2008. ICMLA'08. Seventh International Conference on*. IEEE. 2008, págs. 362-367.

[26] Leonardo Rigutini et al. "Automatic Generation of Crossword Puzzles". En: *International Journal on Artificial Intelligence Tools* 21.03 (2012).

[27] A Sánchez y Melva Márquez. "Hacia un sistema de extracción de definiciones en textos jurídicos". En: *Actas de la 1er Jornada Venezolana de Investigación en Lingüística e Informática* (2005), págs. 1-10.

[28] Gerardo Sierra. "Extracción de contextos definitorios en textos de especialidad a partir del reconocimiento de patrones lingüísticos". En: *Linguamática* 1.2 (2009), págs. 13-37.

[29] Gerardo Sierra. *Extracción de relaciones léxicas para dominios restringidos a partir de contextos definitorios en español*. Inf. téc. México: UNAM, IPN, INAOE, COLMEX, UAM, 2011.

[30] Leon Sterling. *The art of Prolog: advanced programming techniques*. 2nd Rev Ed. MIT press, 1994.

[31] Stephan Walter y Manfred Pinkal. "Automatic extraction of definitions from German court decisions". En: *Proceedings of the workshop on information extraction beyond the document*. Association for Computational Linguistics. 2006, págs. 20-28.

Otras fuentes

[32] Irene Castellón. *Definiciones de corpus anotado*. (Consulta noviembre 2014). URL: http://www.ub.edu/diccionarilinguistica/content/corpus-anotado.

[33] *Documentación de Prolog*. (Consulta abril 2015). URL: http://www.swi-prolog.org/.

[34] *Documentación de Python*. (Consulta noviembre 2014). URL: https://www.python.org/doc/.

[35] *Documentación de XPCE (paquete gráfico para Prolog)*. (Consulta abril 2015). URL: http://www.swi-prolog.org/packages/xpce/UserGuide/.

[36] *Documentación sobre el encoding de MySQL*. (Consulta noviembre 2014). URL: http://dev.mysql.com/doc/refman/5.0/en/charset-connection.html.

[37] *FreeLing*. (Consulta octubre 2014). URL: http://nlp.lsi.upc.edu/freeling/.

[38] Jan Goyvaerts. *Expresiones regulares recursivas*. (Consulta enero 2015). URL: http://www.regular-expressions.info/.

[39] Philip Hazel. *Perl compatible regular expressions*. (Consulta noviembre 2015). URL: http://www.pcre.org/.

[40] *Introducción a las etiquetas EAGLES*. (Consulta agosto 2014). URL: http://nlp.lsi.upc.edu/freeling/doc/tagsets/tagset-es.html.

[41] *Material sobre* backtracking *y* greedy. (Consulta mayo 2015). URL: http://www.codegaia.com/index.php?title=Algoritmia.

[42] *Material sobre unificación*. (Consulta mayo 2015). URL: http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse5.

[43] *Material teórico Introducción al Procesamiento del Lenguaje Natural Facultad de Ingeniería UdelaR*. (Consulta noviembre 2014). URL: https://eva.fing.edu.uy/course/view.php?id=211.

[44] *Mil palabras del Uruguay*. Academia nacional de letras. (Consulta abril 2015). URL: http://www.mec.gub.uy/academiadeletras/Bpalabras/Pp_Palabras.htm.

[45] *Natural Language Toolkit Documentation*. (Consulta enero 2015). URL: http://www.nltk.org/.

[46] *Palabras de 2 y 3 letras*. Asociación uruguaya de scrabble. (Consulta abril 2015). URL: http://www.scrabbel.org.uy/ayuda/dos_y_tres_letras.htm.

[47] Nikita Popov. *The true power of regular expressions*. (Consulta marzo 2015). URL: https://nikic.github.io/2012/06/15/The-true-power-of-regular-expressions.html.

Glosario

ambiguo admite distintas interpretaciones.

análisis morfológico reconocer una palabra y construir una representación estructurada de ella, tomando en cuenta sus morfemas. A cada palabra se le asigna su lema y todas las categorías gramaticales (PoS tags) posibles.

análisis morfosintáctico (**PoS tagging**) tarea de asignar una categoría gramatical (etiqueta) a cada palabra. Esta tarea requiere de un análisis morfológico previo y del contexto de una palabra para determinar la etiqueta correcta, desambiguando entre todas las posibilidades. También recibe el nombre de *análisis léxico*.

análisis sintáctico (**parsing**) consiste en generar un árbol de análisis sintáctico (árbol de parsing) a partir de un texto y una gramática dados, si es posible [1]. Existen distintos tipos de análisis sintácticos, entre los que se destacan *full parsing*, *shallow parsing* y *dependency parsing*. El *full parsing* provee un análisis de la estructura de la oración lo más detallado posible, mientras que el *shallow parsing* es menos detallado, usa distinciones menos finas en cuanto a los tipos de constituyentes, ignorando por ejemplo su estructura interna y su rol en la oración principal. El *dependency parsing* se basa en las relaciones de dependencia entre las palabras (como ser las funciones sintácticas), no existiendo el concepto de constituyentes.

backtracking técnica de diseño de algoritmos aplicable a problemas para los cuales se quiere obtener un conjunto de soluciones o la solución óptima según ciertas condiciones. Esta técnica es una alternativa a la fuerza bruta logrando las mismas soluciones de una manera más eficiente [2].

bootstrapping método de aprendizaje automático semi-supervisado, que necesita solamente un muy pequeño conjunto de entrenamiento etiquetado a mano (seeds o semillas) [3].

[1] En el presente informe se realiza una castellanización del verbo inglés *to parse* como *parsear*, a los efectos de ser más concisos en las descripciones.

[2] Extraído de [41]. Para más información referirse a dicha fuente.

[3] Por más detalles ver apéndice A.

casillero [de un crucigrama] lugar que surge de la intersección de una fila con una columna del tablero.

categoría gramatical caracteriza la función sintáctica de una palabra. Las categorías gramaticales comúnmente utilizadas son: adjetivos, adverbios, determinantes, nombres, verbos, pronombres, conjunciones, interjecciones y preposiciones.

corpus colección de material lingüístico. Para construir un corpus se deben definir las características deseadas: si se trata de material escrito u oral, cantidad de idiomas que presenta (monolingüe vs. multilingüe), el tipo de texto que se trata (por ej.: prensa, literario, científico), el dominio abarcado (por ej.: arte, lingüística), y si se encuentra anotado o no. Un corpus anotado es aquel que cuenta con anotaciones que lo enriquecen lingüísticamente. Estas surgen a partir de un proceso de análisis manual o automático. Las anotaciones pueden ser de distintos niveles: morfológicas, sintácticas, léxico-semánticas, entre otros.

correferencias la resolución de correferencias (o relaciones anafóricas) refiere a encontrar a qué entidad, aparecida anteriormente en el texto, se está haciendo referencia con una palabra o expresión en particular. Por ejemplo al decir "Ayer vi una casa. Su cocina era enorme", vemos que "su" señala implícitamente que la cocina es la de la casa previamente nombrada. También se pueden encontrar casos de mayor complejidad, por ejemplo, en el texto: "El ranking involucra a 323 supermercados y autoservicios, y toma la información que proviene de los propios puntos de ventas", se precisa información semántica para saber que "puntos de venta" refiere a los "323 supermercados y autoservicios".

crucigrama consiste en una cuadrícula (tablero) en donde cada casillero puede estar en blanco (casillero vacío) o en negro (casillero negro), y un conjunto de pistas. Las pistas son pequeños textos que tratan de describir las palabras que deberán colocarse en la cuadrícula, utilizando casilleros vacíos contiguos horizontal o verticalmente (slots) e iniciando en un casillero específico según se indique, insertando un carácter alfabético en cada uno. El crucigrama está completo cuando todos los casilleros en blanco han sido completados.

cruzamiento [algoritmos evolutivos] mezcla de información de dos o más individuos para formar un individuo diferente.

cuantificador [expresión regular] símbolo que especifica la frecuencia con la que un token puede ocurrir. Los cuantificadores más comunes son *?*, *+* y ***.

definendum término que se está definiendo.

definición término que describe a un definendum dado.

expresión regular notación algebraica (fórmula en un lenguaje especial) que caracteriza un conjunto de strings. Sirve tanto para su reconocimiento como para definir un lenguaje formal (gramáticas regulares) [4].

greedy método que consiste en analizar el conjunto de candidatos y tomar en cada etapa una decisión óptima (elegir el mejor candidato). Esta decisión es tomada localmente, o sea sin tener en cuenta las decisiones de etapas pasadas o de etapas futuras [5].

grupo de captura [expresión regular] parte de la expresión regular que se agrupa para poder referenciar luego al texto que reconoce. Por ejemplo, dado el texto "caso" reconocido por la expresión regular `cas(os*)` , `(os*)` captura el substring "o".

grupo nombrado [expresión regular] grupo de captura al cual se le asigna un nombre. Por ejemplo, `(?P<g1> a*)` es un grupo de captura de nombre "`g1`".

grupo recursivo [expresión regular] grupo nombrado que se define en términos de sí mismo, es decir, la subexpresión que lo define contiene una referencia al grupo definido. Por ejemplo, `(?P<g1> a(b|(?&g1)))` es un grupo recursivo que reconoce tiras de a que terminan en b. Este se interpreta como `(a(b|(a(b|...))))`.

individuo [algoritmos evolutivos] cada una de las entidades que representan las soluciones al problema.

inteligencia artificial área multidisciplinaria que intenta imitar la inteligencia humana para crear y diseñar entidades capaces de resolver cuestiones por sí mismas.

inter-annotator agreement porcentaje de coincidencias de las anotaciones realizadas sobre un cierto corpus, por distintos anotadores. Da una idea del nivel de consenso entre dichos expertos sobre cuáles son las etiquetas correctas para dicho corpus.

lema [lingüística] unidad autónoma constituyente del léxico de un idioma. Es una serie de caracteres que forman una unidad semántica y que puede constituir una entrada de diccionario. Por ejemplo el lema de *gatita* es *gato*, y el del verbo *corro* es *correr*.

lenguaje natural lenguaje hablado por las personas (cualquiera de los idiomas del mundo se considera lenguaje natural, en contraste con los lenguajes formales definidos en computación).

machine learning subárea de la inteligencia artificial, que consiste en la construcción y estudio de sistemas que pueden *aprender* a partir de los datos, en vez de sólo seguir

[4] Basado en la definición de Jurafsky y Martin [18]
[5] Extraído de [41]. Para más información referirse a dicha fuente.

instrucciones programadas explícitamente. Se dice que un algoritmo aprende de la experiencia E con respecto a alguna clase de tareas T y medida de performance P, si su performance en tareas de clase T, según la medida P, mejora con la experiencia E.

mutación [algoritmos evolutivos] modificación aleatoria de los cromosomas del padre para obtener un individuo diferente.

parsing análisis sintáctico.

pattern matching acto de chequear una secuencia de tokens dada buscando los constituyentes de algún patrón.

pistas (**claves**) [de un crucigrama] pequeños textos que tratan de describir las palabras que deberán colocarse en un determinado slot.

población [algoritmos evolutivos] conjunto de individuos.

PoS tagger (Part-Of-Speech Tagger) software que lee texto en algún idioma y asigna etiquetas gramaticales (PoS tags) a cada palabra (y otros tokens), como por ejemplo *nombre*, *verbo*, *adjetivo*, etc., aunque generalmente las aplicaciones computacionales usan PoS tags más refinadas como *nombre-plural* [6].

proposición unidad lingüística de estructura oracional, esto es, constituida por sujeto y predicado, que se une mediante coordinación o subordinación a otra u otras proposiciones para formar una oración compuesta.

selección [algoritmos evolutivos] elección de los individuos que sobrevivirán y conformarán la siguiente generación.

slot [de un crucigrama] conjunto de casilleros contiguos horizontal o verticalmente en un tablero, que no contiene casilleros negros.

stop words palabras que son filtradas (descartadas) antes o después del procesamiento de un texto, porque no interesan. Generalmente se trata de una lista de palabras muy frecuentes que no aportan información sustancial (por ejemplo determinantes, y preposiciones). Varían según la tarea que se esté realizando.

string secuencia de símbolos; para la mayoría de las técnicas de búsqueda basadas en textos, un string es cualquier secuencia de caracteres alfanuméricos (letras, números, espacios, tabulaciones, y símbolos de puntuación).

[6] Basado en la definición de `http://nlp.stanford.edu/software/tagger.shtml`.

tablero [de un crucigrama] cuadrícula o matriz de M filas por N columnas, usualmente cuadrada (M = N).

tesauro colección de palabras y definiciones similar a un diccionario, pero que contiene además relaciones semánticas entre sus términos (sinonimia, antonimia, meronimia, hiponimia, etc.).

token instancia de una secuencia de caracteres en un documento particular que se agrupan como una unidad semántica útil para el procesamiento [7].

tokenización [del inglés tokenization] es la tarea de cortar una secuencia de palabras en tokens.

unificación Dos términos (esto es, constantes, variables o términos complejos) *unifican* si son idénticos o si contienen variables que pueden ser instanciadas uniformemente de forma que el resultado final son términos iguales. El mecanismo de unificación chequea si dos términos unifican, y de ser así realiza las sustituciones necesarias para que queden unificados [8].

[7] Texto original de Manning, Raghavan y Schütze [20] "A token is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing".

[8] Para más información referirse al apéndice A.

Apéndice A

Marco teórico ampliado

Procesamiento del lenguaje natural

Es una subdisciplina de la inteligencia artificial. Su objetivo es construir un conjunto de métodos y técnicas eficientes desde un punto de vista computacional para la comprensión y generación de lenguaje natural.

El procesamiento del lenguaje natural abarca el estudio de todos los niveles de análisis lingüístico, entre los que destacamos:

- **Morfología**: estudio de la estructura interna de las palabras.
- **Sintaxis**: estudio de la estructuración (orden y agrupamiento) de las palabras en estructuras mayores.
- **Semántica**:
  - *léxica*: significado de cada palabra
  - *composicional*: significado de la combinación de palabras para obtener significados mayores.

Uno de los mayores desafíos a los que se enfrenta el PLN es el tratamiento de las ambigüedades del lenguaje. En los distintos niveles de análisis se encuentran distintos tipos de ambigüedades, que pueden ser fáciles de discernir para los seres humanos pero no así para las máquinas.

Ejemplo A.1. *Ambigüedades en los distintos niveles de análisis* [1]

[1] Tomado de [43]

Nivel	Ejemplo	Ambgüedad
Morfológico	"Plantamos papas."	¿"plantamos" está en presente o pasado?
Sintáctico	Pedro vio a Juan con el telescopio.	¿Quién tiene el telescopio, Pedro o Juan?
Semántico	La perra de mi vecina.	¿"perra" refiere a una mascota o es una forma de insultar a la vecina?

TABLA A.1: Ejemplos de ambigüedades

El procesamiento de textos suele constar de varias etapas sucesivas, como se muestra en la figura A.1. La entrada consiste en textos, escritos u orales, en lenguaje natural, mientras que la salida depende de lo que se desee realizar. Según el trabajo que se esté desarrollando, pueden hacer falta más o menos etapas.

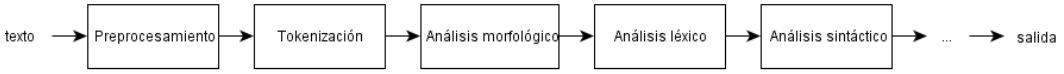

FIGURA A.1: Etapas del PLN

Preprocesamiento

El preprocesamiento abarca todas las tareas iniciales que deben ser desempeñadas antes de poder empezar realmente con las tareas que persiguen directamente el objetivo del trabajo. Varía según la tarea que se desee desempeñar, y puede incluir un proceso en el cual se eliminen las entradas que no cumplen con un nivel mínimo de calidad preestablecido.

Tokenización

Dada una secuencia de caracteres y un texto, la tokenización (*tokenization* en inglés) es la tarea de "cortar" la secuencia en fragmentos, llamados tokens, eventualmente descartando algunos caracteres como ser signos de puntuación. Usualmente se desea que los nombres compuestos estén en el mismo token (por ejemplo "América del Sur" y no "América" "del" "Sur"), y se puede querer también considerar las locuciones como un solo token, lo que puede dar lugar a ambigüedad.

Ejemplo A.2. *Distintas tokenizaciones para "a pesar de"* [2]

[2] Tomado de [43]

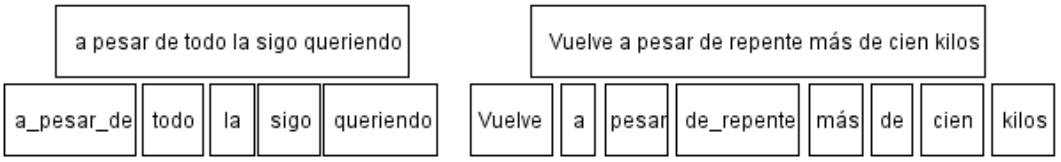

FIGURA A.2: Ejemplos de tokenización

Análisis morfológico

Reconocer una palabra y construir una representación estructurada de ella, tomando en cuenta sus morfemas (subpartes).

Ejemplo A.3. *Análisis morfológico*
La palabra "gatitos" surge de transformar la palabra "gato" en su versión masculina (Masc) plural (Pl), y aplicarle un diminutivo (Dim).

palabra: gatitos
representación estructurada: gato + Masc + Pl + Dim

Análisis léxico

Dado un texto y un conjunto de etiquetas, se desea asignar una categoría gramatical léxica (etiqueta) a cada constituyente. Esta tarea se conoce también como *PoS tagging*, del inglés *Part of Speech* (categoría gramatical) *tagging* (etiquetado). La figura A.3[3] muestra esquemáticamente el proceso de análisis léxico. Existen diversas herramientas, dependientes del idioma trabajado, para realizar automáticamente el análisis léxico de textos, y distintos estándares en cuanto al conjunto de etiquetas a utilizar.

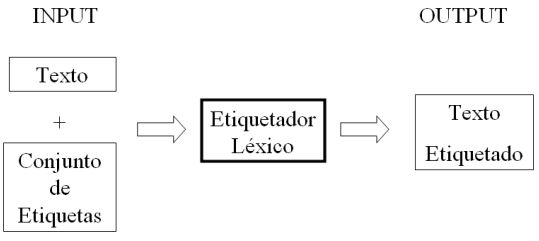

FIGURA A.3: Análisis léxico

Ejemplo A.4. *Análisis léxico de una oración*

En la figura A.4, se muestra el ejemplo anterior usando la herramienta FreeLing, que emplea el conjunto de etiquetas EAGLES (FreeLing muestra el lema de cada palabra en azul y su etiqueta en rojo).

[3]Imagen tomada de [43].

Entrada:	"Yo bajo con el hombre bajo a tocar el bajo bajo la escalera."
Salida:	"Yo/pronombre bajo/verbo con/preposición el/determinante hombre/nombre bajo/adjetivo a/preposición tocar/verbo el/determinante bajo/nombre bajo/preposición la/determinante escalera/nombre ./signo de puntuación".

Yo	bajo	con	el	hombre	bajo	a	tocar	el	bajo	bajo	la	escalera	.
yo	*bajar*	*con*	*el*	*hombre*	*bajo*	*a*	*tocar*	*el*	*bajo*	*bajo*	*el*	*escalera*	*.*
PP1CSN00	VMIP1S0	SPS00	DA0MS0	NCMS000	AQ0MS0	SPS00	VMN0000	DA0MS0	NCMS000	SPS00	DA0FS0	NCFS000	Fp

FIGURA A.4: Análisis léxico

Análisis sintáctico

El análisis sintáctico (*parsing*) consiste en generar un árbol de análisis sintáctico (*árbol de parsing*) a partir de un texto y una gramática dados, si es posible. La idea es agrupar las palabras en constituyentes sintácticos y jerarquizarlos, según las reglas de la gramática elegida. La figura A.5[4] muestra esquemáticamente el proceso de análisis sintáctico. El analizador sintáctico también recibe el nombre de *parser*.

FIGURA A.5: Análisis sintáctico

Ejemplo A.5. *Análisis sintáctico de una oración*
Entrada: "El gato come pescado y bebe agua".
Salida: Ver figura A.6 [5].

[4] Imagen tomada de [43].
[5] Tomada de http://nlp.lsi.upc.edu/FreeLing/demo/demo.php

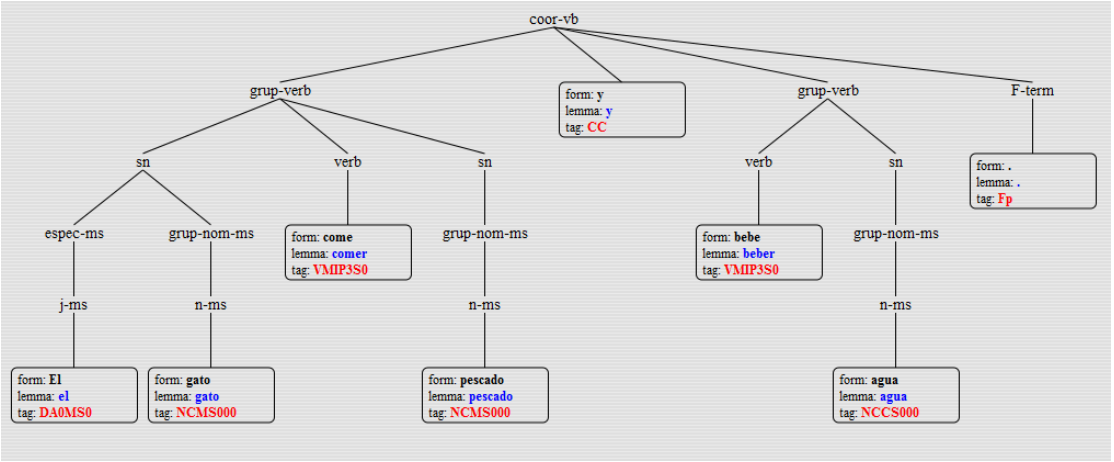

FIGURA A.6: Árbol de parsing de una oración

Gramáticas libres de contexto

La gramática que el analizador sintáctico recibe como entrada puede ser idealmente cualquiera, pero la gramática real de un lenguaje natural es muy compleja, por lo cual se utilizan modelos que la simplifican. Las gramáticas libres de contexto (GLC) son uno de los formalismos que permite dicha simplificación.

Las GLC capturan la noción de constituyente sintáctico y la de orden, y puede verse como generadoras y estructuradoras a la vez.

Características del enfoque GLC:

- Reconocimiento eficiente.
- Poca adecuación lingüística: el conjunto de tiras generadas/aceptadas es adecuado, pero las descripciones estructurales que genera no lo son.
- Puede sufrir problemas de concordancia: “la avión”, “un colina”.
- Presenta problemas de subcategorización: no todas las combinaciones de categorías gramaticales son posibles, y las reglas deben preverlo (ej. sintáctica: Juan quiere salir/ * Juan comió correr; semántica: Juan dijo la verdad, * Juan dijo la mesa. Presencié el incendio, * Presencié la silla).

Entidades con nombre

Una entidad con nombre es cualquier cosa que pueda ser referenciada con un nombre propio.

El proceso de reconocimiento de entidades con nombre (a veces llamado *clasificación de entidades con nombre*) refiere a la tarea combinada de encontrar fragmentos de texto que constituyen nombres propios y clasificarlos según su tipo (con algún conjunto de categorías posibles predefinidas)[6].

Forma de trabajo en PLN

Usualmente en tareas de PLN, dado un problema a trabajar se realizan las siguientes acciones:

1. se recopila un conjunto de textos relevante (corpus)
2. se los divide en corpus de entrenamiento y corpus de testeo
3. se ajusta el modelo trabajando sobre el corpus de entrenamiento
4. se evalúa el modelo sobre el corpus de testeo, nunca antes visto

Bootstrapping

Es un método de aprendizaje automático semi-supervisado. En el contexto de este proyecto, se consideró el enfoque planteado en Hearst [17]. A continuación se describe brevemente el procedimiento.

Inicialmente se descubren algunos patrones por observación, mirando los textos e identificando los patrones y las definiciones que ellos indican.

Luego se ejecuta el siguiente loop hasta alcanzar la condición de parada:

1. Recolectar un conjunto de pares $\prec$ *definición* , *definendum* $\succ$ automáticamente, a partir de los patrones con los que se cuenta.
2. Encontrar los lugares del corpus donde estas definiciones y definendums ocurren próximos sintácticamente y registrar su entorno.
3. Encontrar las similitudes entre estos entornos, suponiendo que estas similitudes se basan en un patrón que indica la relación de definición-definendum, e inferir así el nuevo patrón.
4. Una vez que se obtuvo un nuevo patrón, usarlo para conseguir más instancias de definiciones, e ir al paso 1.

[6] Definición extraída de Jurafsky y Martin [18]

La condición de parada podría ser no encontrar más pares $\prec$ *definición* , *definendum* $\succ$, o alcanzar alguna cantidad de patrones.

Ejemplo A.6. *Aplicación de bootstrapping de patrones*[7]
Inicialmente se descubrió por observación el siguiente patrón (simplificado):

SN {,} especialmente {SN,} {o/y} SN*

es decir, un sintagma nominal, seguido opcionalmente de una coma, la palabra "especialmente" y una lista de uno o más sintagmas nominales separados por comas y finalmente por la palabra "o" o la palabra "y". Este patrón debe contener también la información necesaria para saber que la definición es el primer sintagma nominal en su forma singular, y que los sintagmas nominales sucesivos corresponden a definendums.

Paso 1
Se tiene dentro del corpus el siguiente texto:

"... la mayoría de los países europeos, especialmente Francia, Inglaterra, y España..."

Aplicando el patrón, se obtienen los pares:
$\prec$ *país europeo , Francia* $\succ$
$\prec$ *país europeo , Inglaterra* $\succ$
$\prec$ *país europeo , España* $\succ$

Paso 2
En otra parte del corpus se tiene el siguiente texto:

"... Francia es uno de los países europeos más visitados..."

por lo que se registra el entorno "Francia es uno de los países europeos más visitados".

Paso 3
Tras un análisis, se identifica al nuevo patrón:

"SN es uno de los SN"

donde el definendum es el primer SN y la definición es el segundo SN en su forma singular.

[7] Tomado de Hearst [17] y adaptado.

Paso 4

Al analizar el texto:

"Italia es uno de los principales países productores de vino"

utilizando el nuevo patrón, se encuentra el par:

$$\prec \textit{principal país productor de vino , Italia} \succ .$$

Y se sigue iterando dentro del loop, hasta alcanzar la condición de parada.

Medidas de evaluación

La *precision*[8], el *recall*, y la *medida F* son medidas usualmente utilizadas para evaluar resultados en diversas áreas de PLN.

Para poder medir qué tan bien funciona un algoritmo, es necesario saber qué tan acertada fue su salida respecto a los resultados esperados. Para ello se suele clasificar los objetos del universo considerado del siguiente modo:

- *True Positive* o verdadero positivo (**TP**): son aquellos objetos recuperados acertadamente por el algoritmo (debían ser recuperados).
- *False Positive* o falso positivo (**FP**): son aquellos objetos que el algoritmo recuperó, pero no debían ser recuperados.
- *False Negative* o falso negativo (**FN**): son aquellos objetos que el algoritmo no recuperó cuando debería haberlo hecho.
- *True Negative* o verdadero negativo (**TN**): son aquellos objetos que el algoritmo acertadamente no recuperó.

En nuestro caso, el universo considerado es un corpus anotado, donde se indican todas las definiciones existentes. Considerando la salida de un algoritmo de extracción de definiciones correspondiente a ese corpus, podemos clasificar cada definición en una de estas 3 categorías:

- *True Positive* (**TP**): definición hallada por el algoritmo que realmente es una definición.

[8] Este término suele traducirse al español como "precisión", pero genera algunos malentendidos con el término "accuracy", otra medida utilizada en PLN, que también se traduce como "precisión".

- *False Positive* (**FP**): definición hallada por el algoritmo que no es una definición real.
- *False Negative* (**FN**): definición real que no fue hallada por el algoritmo.

Observar que en nuestro caso, dado que no hay objetos individuales sino un texto continuo, no aplica la noción de *True Negative*.

También se trabaja con corpus no anotados, pudiendo obtener las medidas de TP y FP, pero no así las de FN.

La **precision** (P) se puede ver como la cantidad de objetos correctos recuperados por el algoritmo, sobre la cantidad de objetos obtenidos por él ($P = TP/(TP + FP)$).

El **recall** (R) se puede ver como la cantidad de objetos correctos obtenidos por el algoritmo sobre la cantidad de objetos correctos del universo ($R = TP/(TP + FN)$).

Finalmente la **medida F** es la media armónica de las dos anteriores: $Medida\_F = 2P.R/(P + R)$.

Unificación

Términos en Prolog

Existen tres tipos de términos:

1. *Constantes*. Pueden ser átomos, que se representan iniciando con minúscula (como *vicente*) o bien números (como *24*).
2. *Variables*. Siempre se representan iniciando con una mayúscula o guión bajo (como *X* , *_Z3* y *Lista*).
3. *Términos complejos*. Tienen la forma de *functor(término_1,...,término_n)*, o forma de listas *[término_1,...,término_n]*.

Definición de unificación

Una primera aproximación a la definición de unificación es la siguiente:

Dos términos *unifican* si son el mismo término o si contienen variables que pueden ser instanciadas uniformemente con términos de forma que el resultado final son términos iguales.

Por ejemplo, los términos *mia* y *mia* unifican, porque son el mismo término atómico, y análogamente sucede con los términos *42* y *42*, porque son el mismo número. Asimismo, los términos *mujer(mia)* y *mujer(mia)* unifican, porque son el mismo término complejo. Sin embargo, los términos *mujer(mia)* y *mujer(ana)* no unifican porque no son el mismo y ninguno de ellos contiene variables que puedan ser instanciadas para hacer que queden iguales.

Además, los términos X y *mia* unifican, porque la variable X se puede instanciar con el valor *mia* haciéndolos iguales. Análogamente sucede con los términos *mujer(X)* y *mujer(mia)*. Sin embargo, los términos *ama(vicente,X)* y *ama(X,mia)* no unifican, ya que es imposible encontrar una instanciación de X que los transforme en términos iguales: instanciar X con *vicente* nos da los término *ama(vicente,vicente)* y *ama(vicente,mia)*, que no son iguales, e instanciar X con *mia* nos da los términos *ama(vicente,mia)* y *ama(mia,mia)* que tampoco son iguales.

Usualmente no solo estamos interesados en saber si dos términos unifican, sino que también queremos saber cómo deben instanciarse las variables para hacer los términos iguales. Cuando Prolog unifica dos términos, realiza todas las instanciaciones necesarias luego de lo cual todos los términos quedan iguales. Esta funcionalidad, junto con el hecho de que podemos construir términos complejos (estructurados recursivamente, por ejemplo) hace de la unificación un poderoso mecanismo de programación. [9]

[9] Esta sección está basada en [42]. Para una definición más exacta de unificación y para más detalles, referirse a dicha fuente.

Apéndice B

PCRE generada

La expresión utilizada es diferente para cada patrón y se crea a partir de éste. Para generarla tenemos en cuenta los elementos del patrón parser (ver capítulo 4) de la siguiente manera:

1. *parámetro1* = ≺DEF≻ o ≺DFM≻:
   creamos un grupo llamado *definicion* o *definendum* respectivamente, que reconozca en el árbol de parsing todo lo contenido en el tipo de constituyente indicado en *parámetro3*. En este caso *parámetro2* no se utiliza en la expresión regular, sino que, después de encontrado el par ≺ *definición* , *definendum* ≻ , se lo validará verificando que cumpla con las restricciones del *parámetro2*. Este grupo se trata de un grupo nombrado, para obtener a partir del nombre (*definicion* o *definendum*) el resultado de dicho grupo, es decir, la definición o el definendum buscados.

   Notamos que existen definiciones (y definendums) que contienen constituyentes anidados, por lo que fue necesario dividir la definición en dos partes: ≺DEF≻ (para el constituyente principal) y ≺DEFC≻ (para el constituyente que complementa al anterior), de forma análoga, se utilizan ≺DFM≻ y ≺DFMC≻ para el definendum.

2. *parámetro1* = ≺DEFC≻ o ≺DFMC≻:
   se trata de un grupo opcional, es decir, puede aparecer o no en el texto.
   Creamos un grupo similar al anterior con los nombres correspondientes. Este tipo de elemento representa a los complementos de la *definición* o *definendum* mencionados anteriormente. Una vez aplicada la expresión regular, los fragmentos de texto reconocidos por los grupos *definicion* y *definendum* son unidos con el de sus grupos

complementarios, en caso de haberlos, para así formar la definición (definendum) completo.

3. ⟨*parámetro1,parámetro2,parámetro3*⟩ = ⟨*palabra,lema,tag*⟩:
   creamos un grupo llamado *word* que reconozca la palabras cuyo PoS tag unifique con ⟨*palabra,lema,tag*⟩

En los dos primeros casos, se utilizarán grupos de captura recursivos que explicaremos más adelante. De esta forma, se itera sobre cada elemento del patrón y se van concatenando las subexpresiones generadas por cada uno, para así formar la expresión final encargada de extraer el par ≺ *definición* , *definendum* ≻ .

Explicaremos la expresión regular generada por medio de un ejemplo.

Dado el patrón parser

[⟨≺DFM≻,_,sn⟩, ⟨_,ser,Verbo⟩, ⟨≺DEF≻,_,sn⟩]

la expresión regular asociada es

sn_(?P<definendum>[expr_grupos_internos])
expr_aux ser Verbo expr_aux
sn_(?P<definicion>[expr_grupos_internos])

donde $?P < definendum >$ reconoce el texto correspondiente al *definendum*, $?P < definicion >$ reconoce el texto de la *definición* y la subexpresión expr_grupos_internos reconoce los constituyentes internos de la definición y el definendum (cada constituyente está delimitado por paréntesis rectos).

Ya que ambos grupos ($?P < definendum >$ y $?P < definicion >$) tienen composiciones similares, analizaremos solamente uno de ellos.

La representación gráfica del grupo $?P < definendum >$ es la siguiente:

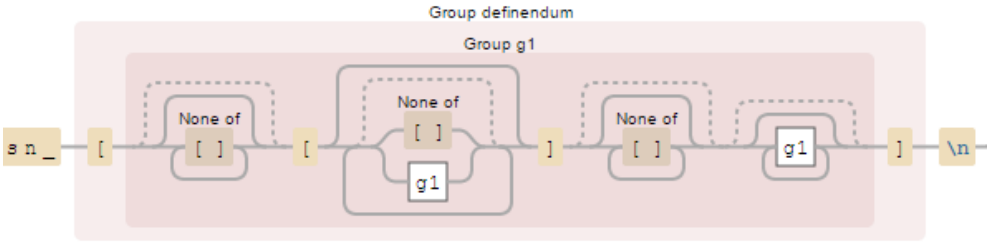

FIGURA B.1: PCRE: Representación gráfica del grupo *definendum*

dada por la expresión

sn_(?P<definendum>[expr_grupos_internos])

Notamos que el grupo *definendum* busca reconocer todo el texto que constituye el sintagma nominal *sn* (desde el paréntesis recto inicial hasta el final). Debido a que dentro del sintagma nominal puede haber otros constituyentes (también delimitados por paréntesis rectos), se crea el grupo *g1*, encargado de reconocerlos, y se repite tantas veces como constituyentes "hijos" tenga el sintagma. *g1* realiza un balanceo de los paréntesis, para reconocer a cada constituyente. Estos últimos, pueden también contener otros en su interior, por lo que *g1* es recursivo.

Consideremos el siguiente árbol de parsing:

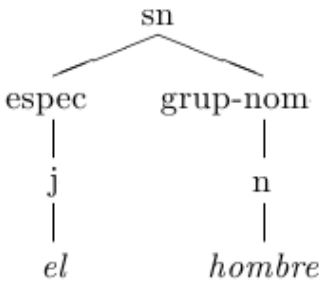

FIGURA B.2: PCRE: Árbol de parsing de "*El hombre*"

El grupo *definendum* reconoce el sintagma nominal, el primer grupo *g1* todo lo contenido en *espec*, el *g1* interior a éste lo contenido en *j* , el grupo *g1* análogo *grup-nom*, etc., como se muestra en la figura B.3.

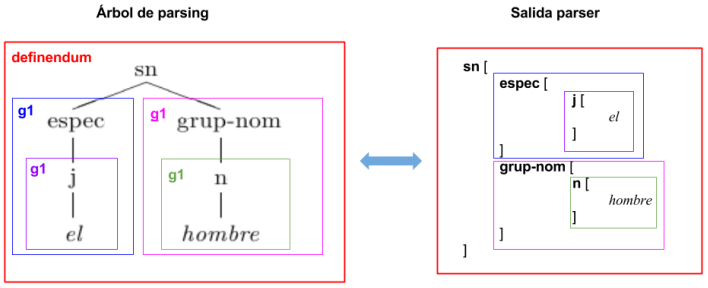

FIGURA B.3: PCRE: Correspondencia entre el árbol y la salida del parser

Optimización de la expresión regular

Otra característica importante de la expresión generada es la utilización de optimizaciones. Al emplear grupos recursivos, tenemos cuantificadores anidados. Esto casi siempre genera *backtracking catastrófico*, es decir, el cómputo de la expresión es infinito ya que hay infinitos puntos de backtracking.

Para solucionar esto, al anidar operadores de repetición, debemos asegurarnos de que haya una sola manera de reconocer cada texto.

Realizamos entonces dos tipos de optimizaciones:

Utilización de grupos atómicos

Un grupo atómico es un grupo que, luego de encontrar una coincidencia, automáticamente descarta todos sus puntos de backtracking. Es decir, si dentro de un grupo existe más de una alternativa, una vez que una de ellas tuvo éxito, ya no se consideran las restantes.

Consideremos el texto *aab*. La expresión (a|aa)b lo reconoce: aa coincide con *aa* y b con *b*. Por otro lado, la expresión con un grupo atómico (?>a|aa)b no reconoce al texto: a coincide con *a* y se eliminan los puntos de backtracking, luego b no coincide con *ab*.

Utilización de cuantificadores posesivos

Los cuantificadores posesivos son una forma de prevenir que se prueben todas las permutaciones, lo cual es principalmente útil por razones de eficiencia.

Un cuantificador posesivo repite el token tantas veces como sea posible, sin liberar tokens reconocidos a medida que se realiza backtracking. Con un cuantificador posesivo, se toma un enfoque de "todo o nada", lo que permite que la expresión falle más rápido.

Consideremos el texto *aaab*. La expresión a*ab lo reconoce: a* coincide con *aa* y ab con *ab*. Sin embargo, la misma expresión con un cuantificador posesivo a*+ab no lo reconoce: a*+ coincide con *aaa* y se descartan todos los puntos de backtracking, luego ab no coincide con *b*.

Apéndice C

Implementación con backtracking

Recordamos que la generación de crucigramas utilizando backtracking se divide en dos etapas: generación de la máscara y asignación de palabras.

Generación de la máscara

En esta etapa, para ubicar los cuadrados negros, se tienen en cuenta los siguientes criterios de calidad, que se van combinando:

1. *MaxLargos*: máxima cantidad de slots largos.
2. *MaxNegros*: máxima cantidad de casilleros negros.
3. *MaxCortos*: máxima cantidad de slots cortos.

MaxLargos evita que haya demasiadas palabras largas, *MaxCortos* es análogo con las palabras cortas, y *MaxNegros* evita que el crucigrama quede excesivamente pequeño.

Estos criterios se construyeron tomando en cuenta los lineamientos de qué características son deseables en un crucigrama, encontrados en la bibliografía sobre generación de crucigramas automáticos.

Con el fin de obtener una mayor variedad de crucigramas finales, no se impone que se cumplan los tres criterios simultáneamente, sino que se exige el cumplimiento del criterio 1, lo que genera una determinada cantidad de casilleros negros. Si esa cantidad de casilleros supera *MaxNegros*, se devuelve el tablero tal como quedó luego de cumplir con el criterio 1, sin cumplir con el 2 ni verificar el 3. Si no, se agregan más casilleros negros respetando *MaxNegros* y sin superar la cantidad de slots cortos permitidos (*MaxCortos*).

La restricción de la cantidad máxima de casilleros negros establece un máximo global (cantidad total de casilleros negros en todo el tablero) y un máximo de casilleros negros por fila (para que no suceda que una fila contenga una cantidad excesiva de casilleros negros). La ubicación de los casilleros negros en la máscara se realiza de forma aleatoria, respetando las restricciones, para obtener distintos tableros en cada ejecución.

Los valores de los parámetros de calidad y la clasificación de los slots (en largos, cortos y medianos), fueron fijados de manera heurística, luego de realizar pruebas con distintas combinaciones, eligiendo aquella que le daba una mejor apariencia a la máscara incompleta y al crucigrama final.

El algoritmo primero impone el criterio 1, revisando uno a uno los slots horizontales, y en caso de que sean slots largos, se agregan casilleros negros de forma de partir al slot en dos slots no largos. Para ello se elige un casillero del slot, que esté cercano al centro del mismo[1], de forma aleatoria y luego se siguen agregando casilleros negros contiguos a este si es necesario, hasta satisfacer la restricción. El elegir los casilleros contiguos permite no generar slots muy cortos. Luego se colocan casilleros negros adicionales, para ampliar la variedad de máscaras, respetando las restricciones de cantidad máxima de casilleros negros y de slots cortos. Esto se hace recorriendo las filas del tablero, definiendo una cantidad aleatoria de casilleros negros a colocar en cada una (que no sobrepase el máximo por fila ni el máximo total), y eligiendo aleatoriamente los casilleros que se pintarán de negro entre una lista de candidatos (aquellos que no violan el criterio 3).

A continuación, presentamos el pseudocódigo de esta etapa:

Algoritmo 4 generar premascara

```
Entrada: N, max_casilleros_negros_por_fila
Salida: Tablero
  generar_tablero(N,Tablero), % Genera un tablero en blanco de tamaño NxN.
  Para cada S slot largo de Tablero :
    elegir_y_setear_casilleros_negros(S),
  fin para
  Para cada F fila de Tablero :
    CNF = random([0..max_casilleros_negros_por_fila]),
    candidatos(F,ListaCandidatos),
    colocar_casilleros_negros_aleatorios(ListaCandidatos, CNF)
  fin para
```

[1] Debe estar a una distancia máxima del casillero central de un $\frac{1}{6}$ del largo del slot (se definió de forma heurística).

Asignación de palabras

Iteración 1

En la primera iteración se implementó un algoritmo cuya idea central es elegir la forma de recorrer los slots para maximizar las posibilidades de éxito y encontrar los puntos de fallo lo antes posible, para minimizar la cantidad de trabajo innecesario.

Para ello, dado un slot a rellenar, se prueba una a una con las palabras compatibles, y se calcula cuál de sus letras - dentro de las que intersectan con slots vacíos - opone la mayor dificultad para encontrar una palabra que la cruce, obteniendo así la compatibilidad de cada letra. Entonces en el siguiente paso se va a llenar ese slot de mayor dificultad, probando con una lista de palabras candidatas que están ordenadas según su compatibilidad, repitiendo el proceso descrito arriba. Si en algún momento este proceso falla, por no encontrar ninguna palabra que satisfaga al slot, se pone un casillero negro (probando uno a uno con todos los casilleros del slot, mediante backtracking), generando nuevos slots de largo menor, y se llama recursivamente al procedimiento para el próximo slot vacío de la lista.

Esta idea de realizar una selección de las palabras se basó en los artículos leídos, que señalaban que esta era la forma más conveniente de recorrer el espacio de soluciones.

El algoritmo se describe a continuación.

Algoritmo 5 asignar palabras

```
Entrada: Tablero, Palabras
Salida: Tablero
  Slot = primer slot vacío de la lista
  asignar_palabra(Slot)
  si asignar_palabra falló :
    poner_casillero_negro(Slot,NuevosSlots) % aquí se genera un punto de backtracking
    agregar_a_slots(NuevosSlots)
    asignar_palabras(siguiente slot vacío de la lista)
  fin si
```

Algoritmo 6 asignar palabra

```
Entrada: Slot, Palabras
Salida: Slot
  obtener_lista_palabras_compatibles(Lista_Compatibles, Slot)
  si Lista_Compatibles es vacía :
    falla % se vuelve al último punto de backtracking
  si no
    Palabra = get_palabra(Lista_Compatibles) % aquí se genera un punto de backtracking
    colocar(Palabra,Slot)
    % calcular ganancia del tablero
    Para cada Letra en Slot :
      si Letra intersecta con un slot vacío :
        calcular_compatibilidad(Letra)
      fin si
    fin para
    % Nos enfocamos en la letra de menor compatibilidad (más problemática)
    Letra_Problemática = Letra / Letra.compatibilidad es mínima
    asginar_palabra(Slot que intersecta con letra problemática)
  fin si
```

Iteración 2

Se observó que el algoritmo utilizado en la iteración 1 era muy ineficiente, por lo que se decidió dejar que el backtracking se encargara de forma no asistida de ir recorriendo el espacio de soluciones, es decir, ya no se calcula la ganancia de una palabra sino que se la asigna al tablero y se sigue con el próximo slot de la lista. El algoritmo *asignar palabras* queda igual, por lo que aquí se detalla solamente el algoritmo *asignar_palabra*.

Algoritmo 7 asignar palabra

```
Entrada: Slot, Palabras
Salida: Slot
  obtener_lista_palabras_compatibles(Lista_Compatibles, Slot)
  si Lista_Compatibles es vacía :
    falla % se vuelve al último punto de backtracking registrado
  si no
    Palabra = get_palabra(Lista_Compatibles) % aquí se genera un punto de backtracking
    colocar(Palabra,Slot)
    % se sigue con el próximo slot vacío de la lista.
    Siguiente_Slot = get_slot(LSlotsVacios)
    asginar_palabra(Siguiente_Slot)
  fin si
```