Universidad de Zaragoza

Departamento de Informática e Ingeniería de Sistemas

Aplicación de las 2-estructuras a las gramáticas del lenguaje humano y representación gráfica de ambas

Proyecto Fin de Carrera

Ingeniería en Informática

Autor: Daniel Larraz Hurtado

Directora: Elvira Mayodormo Cámara

Centro Politécnico Superior

Agosto de 2010

Curso 2009-2010

Agradecimientos

Este trabajo representa el fin de una etapa en mi vida. Durante ella he tenido la gran suerte de convivir con un grupo de compañeros excepcionales con los que he compartido miles de momentos innolvidables. Su compañía ha hecho del camino recorrido un auténtico placer.

No voy a intentar nombrarles a todos, pues son muchos y existe el peligro de omitir a alguno. Confio en que ellos sepan quiénes son. Sin embargo, estoy seguro de que comprenderan el hecho de que haga una excepción y mencione de manera especial a mi gran amigo, y compañero de prácticas durante casi toda la carrera, Santiago. Durante todo este tiempo Santiago ha sido para mí un referente por su pasión y por su buen hacer tanto en el ámbito académico como en el personal. Le doy mil gracias por su fantástica compañía a largo de estos cinco años.

Por motivos similares a los esgrimidos previamente tampoco voy a hacer una lista de profesores, de los que en gran parte guardo muy buen recuerdo y con los que he aprendido mucho. No obstante, debo agradecer la confianza, la libertad y el apoyo que he recibido por parte de Elvira Mayordomo para desarrollar este proyecto.

Por último, y no por ello en menor grado de importancia sino todo lo contrario, no tengo palabras para expresar mi gratitud por todo lo que les debo a mis padres, Pedro y Asun, y mi novia, Natalia, que han estado a mi lado compartiendo todas mis alegrías y penas y haciendo de mí lo que soy. Sin ellos, el inminente ingeniero en informática que suscribe estas líneas no sería nada.

A ellos les dedico este trabajo.

Aplicación de las 2-estructuras a las gramáticas del lenguaje humano y representación gráfica de ambas

Resumen

La teoría de las 2-estructuras [5] proporciona una infraestructura matemática para la descomposición y la transformación de grafos. Se trata de un formalismo muy potente y robusto que permite representar múltiples grafos en una sola estructura algebraica, una 2-estructura, y derivar de ella una descomposición única en 2-estructuras más simples. En este proyecto se ha llevado a cabo su estudio con dos finalidades:

- El diseño y la implementación de un paquete de software que sistematice el análisis, la transformación y la visualización de las principales estructuras involucradas en la teoría de las 2-estructuras.
- La investigación y el desarrollo de posibles aplicaciones de las 2-estructuras a las gramáticas usadas en el procesamiento del lenguaje humano (lenguaje natural).

El lenguaje natural es casi en cualquier aspecto más complejo de lo esperado [6]. La sintaxis de muchos idiomas incluye reglas gramaticales que son sensibles al contexto, fenónemos cuyo procesado está muy lejos de tener soluciones eficientes (recordemos que los compiladores de lenguajes de programación sólo procesan un subconjunto muy simple de las gramáticas completamente libres del contexto y que el procesado de gramáticas sensibles al contexto es en general inviable). Las gramáticas suavemente sensibles al contexto (*Mildly Context Sensitive Grammars*, MCSG) pretenden capturar la sintaxis del lenguaje natural y conseguir su procesado eficiente [7,9].

Entre los lenguajes que describen estas gramáticas encontramos una subclase de gran interés por los siguientes tres motivos. Los lenguajes que contiene capturan un amplio espectro de las dependencias del lenguaje natural, son reconocibles en tiempo polinómico y existen cuatro formalismos independientes entre sí que los generan [8]. Son los lenguajes descritos por las grámaticas de adjunción de árboles (*Tree Adjoining Grammars*, TAG), las gramáticas de núcleo (*Head Grammars*, HG), las gramáticas lineales de índices (*Linear Indexed Grammars*, LIG) y las gramáticas categoriales combinatorias (*Combinatory Categorial Grammars*, CCG).

En este trabajo se presentan dos resultados producto de la investigación sobre la aplicación de las 2-estructuras a algunas de las gramáticas mencionadas:

- Una extensión de las HG que asocia explícitamente un árbol derivado a las cadenas generadas apoyándose en las bases de las 2-estructuras.
- Un algoritmo que genera una gramática TAG a partir de una frase con dependencias anidadas y cruzadas (las capturables por el formalismo).

Índice

1 Introducción

En este capítulo se describen los elementos fundamentales en que se enmarca el proyecto como pueden ser los objetivos, el alcance, el trabajo previo en que se apoya, el contexto en que se desarrolla o las herramientas que se han utilizado para su realización. Además se indica de forma abreviada el contenido y distribución del resto de capítulos de la memoria.

1.1. Objetivos y alcance

Los objetivos de este proyecto pueden agruparse y resumirse en la consecución de dos metas.

La primera meta del proyecto consiste en el diseño e implementación de un paquete de software que cumpla los siguientes requisitos:

- Permita la creación, la consulta de propiedades y la transformación de las principales estructuras definidas en tres de los papers [1,2,4] que cubren el estudio fundamental sobre la teoría de las 2-estructuras.
- Dibuje gráficamente las representaciones propuestas en los mencionados papers.
- Integre los anteriores componentes en un programa interactivo que permita la manipulación y el almacenamiento permanente de las estructuras.

La segunda meta tiene como propósito la investigación y el desarrollo de posibles aplicaciones de las 2-estructuras a las gramáticas suavemente sensibles al contexto (*Mildly Context Sensitive Grammars*, MCSG) utilizadas en el procesamiento del lenguaje natural. Para ello se toma como punto de partida los siguientes dos subobjetivos:

- Utilización de la herramienta construida para estudiar formas de representación conjunta del orden de escritura y el orden de generación de los terminales de cadenas derivadas de una MCSG, empleando en las pruebas un conjunto suficientemente representativo de ejemplos de gramáticas. El interés de esta tarea viene dado por el hecho de que, al contrario de lo que ocurre con las gramáticas independientes el contexto (*Context Free Grammar*, CFG), en algunas MCSG no se puede deducir de los árboles de derivación las reglas aplicadas en la derivación.
- Establecimiento de conjeturas y/o resultados que permitan fortalecer la formalización de las MCSG mediante el uso de las 2-estructuras. Este último subobjetivo corresponde a un reto actual de investigación del procesamiento del lenguaje natural [6]. En este proyecto se pretende obtener resultados iniciales o al menos conjeturas prometedoras.

Debido a la naturaleza innovadora de la segunda meta, en el transcurso del trabajo se fueron definiendo y perfilando nuevos subobjetivos (derivados del planteamiento inicial) en función de los resultados obtenidos. Ello dio lugar al desarrollo de dos líneas de trabajo:

- La formulación de una extensión de las grámaticas de núcleo (*Head Grammars*, HG) que asocia explícitamente un árbol derivado a las cadenas generadas usando las operaciones de reescritura del formalismo original. La motivación es conseguirlo apoyándose en la robusta teoría de las 2-estructuras.

- El diseño de un algoritmo que dados el orden de escritura y el orden de generación de los terminales de una cadena perteneciente a un lenguaje generado por una MCSG, obtiene un árbol derivado que es compatible con una gramática de adjunción de árboles (*Tree Adjoining Grammars*, TAG). Este método que podría parecer trivial aplicado a los lenguajes generados por una CFG, no lo es tal en las MCSG debido a la existencia de dependencias cruzadas entre terminales.

1.2. Trabajo previo

La teoría de las 2-estructuras fue introducida por Ehrenfeucht y Rozenberg en 1990. Para la elaboración de este proyecto se ha trabajado principalmente con [1,2] donde se establecen los fundamentos de la teoría y con [4], que requiere de lo explicado en [3], donde se profundiza sobre una subclase de 2-estructuras de gran interés, las T-estructuras. De este último paper es de especial importancia la relación de las T-estructuras con la representación de árboles ordenados en el que está basada la extensión de las HG planteada en este trabajo.

Existe un libro [5] que recopila y amplia el contenido de los papers sobre 2-estructuras presentes hasta su publicación.

En cuanto al procesamiento del lenguaje natural y las MCSG se recomienda [6] y [7] donde se puede obtener una introducción y una panorámica general de ambos temas. También es destacable [8] donde se demuestra cómo las TAG, las HG, las gramáticas lineales de índices (*Linear Indexed Grammars*, LIG) y las gramáticas categoriales combinatorias (*Combinatory Categorial Grammars*, CCG) generan la misma clase de lenguajes, es decir, son débilmente equivalentes entre sí. De ahí se utiliza en este documento la definición de las HG, que es equivalente a la dada por Pollard originalmente en [10] pero que resuelve ciertos problemas notacionales.

En [9] se da una visión de la no-independencia del contexto de los lenguajes naturales y se toma para esta memoria los lenguajes propuestos como ejemplos representativos de lenguajes suavemente sensibles al contexto.

Sobre el procesamiento de dependencias anidadas y cruzadas se ha encontrado algún paper como [15] pero toma un enfoque distinto a la aproximación tomada en este trabajo. En él no se pretende construir árboles de derivación sino solamente reconocer con un autómata palabras con dependencias anidadas y cruzadas.

En [11,12] se comenta la ausencia de un árbol derivado asociado a las HG. En [11] se propone una forma de asociar un árbol de derivación (en las MCSG éste se distingue de un árbol derivado) y en [12] se propone informalmente árboles derivados para mostrar ciertas 'equivalencias fuertes' entre las HG y las TAG. Sin embargo, no se llega a resultados concluyentes.

Para la visualización gráfica de los diferentes tipos de árboles del paquete de software construido, se ha utilizado la implementación mejorada descrita en [14] del algoritmo de posicionamiento de nodos propuesto por Walker en [13].

1.3. Contexto

El desarrollo de este proyecto se enmarca dentro del Área de Lenguajes y Sistemas Informáticos del Departamento de Informática e Ingeniería de Sistemas (DIIS) bajo la dirección de Elvira Mayordomo Cámara. Ha sido realizado desde mediados de abril hasta principios de septiembre de 2010.

Con este proyecto se pretende contribuir en la línea de investigación sobre MCSG y 2-estructuras mediante las siguientes dos aportaciones:

- La construcción de una herramienta que facilite el desarrollo de futuros trabajos que se fundamenten en la utilización de la teoría de las 2-estructuras.
- El establecimiento de resultados prometedores que relacionen las 2-estructuras y las MCSG.

También es relevante mencionar que el Lenguaje Natural, importante disciplina de la Informática, no se menciona a penas en los contenidos del plan de estudios de Ingeniería en Informática, por lo que ha sido necesario un esfuerzo adicional de estudio básico del tema.

1.4. Métodos, técnicas y herramientas

En la primera fase del proyecto se invirtió el tiempo en el estudio de la teoría de las 2-estructuras. A medida que se iba avanzando se empezaron a plantear los primeros modelos del universo del discurso usando una metodología orientada a objetos. Después se hizo un diseño pensando en las formas más eficientes de representar las estructuras teniendo en cuenta los métodos algorítmicos de transformación de una

estructura en otra. Tras ello se abordó el diseño de la biblioteca de visualización y luego el análisis y el diseño del programa interactivo.

Para la implementanción se decidió utilizar la plataforma de desarrollo Java SE 6.0 [17]. Se eligió dicha tecnología por ser una herramienta madura, documentada, gratuita, pensada para el desarrollo de aplicaciones de escritorio multiplataforma y con un amplio abanico de bibliotecas predefinidas y externas.

Se contempló diversas bibliotecas para el dibujado de grafos como JGraph [18] y JUNG [19], ya que una 2-estructura tiene la misma representación que un grafo completo con sus aristas coloreadas. Sin embargo, las peculiaridades de la representación de la descomposición de una 2-estructura provocó que se eligiera hacer el dibujado 'manualmente' con la biblioteca predefinida de gráficos 2D de Java.

Durante el desarrollo se utilizó el entorno de programación Eclipse [20] y se mantuvo una copia actualizada del paquete de software en un repositorio remoto con Subversion [21].

En la siguiente fase del proyecto se llevó a cabo la lectura de diverso material relacionado con el procesamiento del lenguaje natural y las MCSG. Después se empleó la herramienta construida para el estudio de aplicaciones de las 2-estructuras a las MCSG. Según los resultados obtenidos se fueron refinando los objetivos de la investigación hasta obtener el trabajo presentado, incluyendo los dos resultados relacionados con las gramáticas HG y las gramáticas TAG y la implementación e integración de los algoritmos correspondientes dentro del paquete interactivo.

1.5. Organización de la memoria

El escrito se estructura en dos partes: memoria y anexos. En la memoria se sintetiza las aportaciones realizadas por el autor. En los anexos se completa el trabajo principal y se presenta el trabajo previo en el que se fundamenta.

El contenido de los capítulos de la memoria es el siguiente. En el capítulo 2 se documenta el análisis, el diseño y la implementación del paquete de software desarrollado. En el capítulo 3 se presenta la extensión de las HG que asocia explícitamente un árbol derivado a las cadenas generadas. En el capítulo 4 se describe el algoritmo para generar una gramática TAG a partir de una frase con dependencias. En el capítulo 5 se resume el trabajo realizado, se da una valoración personal del mismo y se relaciona con las posibles formas de continuarlo.

A continuación se describe el contenido de los anexos. En el anexo A se presenta los conceptos básicos de la teoría de las 2-estructuras. En el anexo B se da una breve introducción al procesamiento del lenguaje natural y se define las gramáticas TAG y las HG que son utilizadas en el trabajo principal. En el anexo C se explica la correspondencia de una gramática TAG con un árbol de dependencias, la estructura

en la que se basa el algoritmo presentado en el capítulo 4. En el anexo D se muestra una implementación en pseudocódigo del algoritmo del capítulo 4. En el anexo E se da una guía de instalación y uso del entorno de análisis y visualización desarrollado dentro del paquete software.

.

2 Desarrollo del paquete de software

En este capítulo se comenta brevemente el modelo de datos de las 2-estructuras[1], las principales decisiones de diseño tomadas para implementarlas, los controles gráficos desarrollados para visualizarlas y la arquitectura del programa interactivo. Para tener una panorámica del funcionamiento del programa se recomienda consultar el manual de usuario[2].

2.1. Análisis y diseño de las 2-estructuras

En la ilustración 2.1 se muestra un diagrama de clases con la representación de las principales estructuras y sus relaciones[3]. En general todas ellas son clases genéricas y admiten ser parametrizadas con cualquier tipo de dato en cuanto a los elementos contenidos y las etiquetas utilizadas.

Cuando se analizó las 2-estructuras, la primera intuición llevó a pensar en modelar las 2-estructuras *etiquetadas* como una especialización de las 2-estructuras. Sin embargo, esto no es posible debido a que algunas operaciones heredadas necesitan redefinir su signatura. Por ejemplo, existe una operación de sustitución de un elemento de una 2-estructura por otra 2-estructura. La operación necesita de un argumento que especifique qué clases de equivalencia de la primera son equivalentes en la segunda (si hay alguna). En su versión etiquetada sigue estando definida la operación pero ya no necesita de dicho argumento porque la información está implícita al estar las aristas (y transitivamente sus clases) etiquetadas. Por tanto, la solución tomada fue usar la clase abstracta `AbstractTwoStructure` donde se implementan las operaciones en común y resolver las discrepancias mediante el uso de subclases.

El razonamiento anterior se extiende a una *labeled tree family*, la estructura resultante de descomponer una 2-estructura. En el diagrama esta relación de construcción de una estructura a partir de la otra está expresada con una dependencia esteriotipada `generate`. Como la operación funciona en ambos sentidos la dependencia es bidireccional.

En cuanto al diseño de la solución de las 2-estructuras y las labeled tree families, destacar que ambas se componen de un dominio (que hace de diccionario entre los elementos de tipo genérico utilizados y su representación interna como índices) y de una implementación *base* (basada en dichos índices). El uso de índices mejora la eficiencia de las operaciones de conversión entre estructuras, al no tener que acceder constantemente a un diccionario, a costa de aumentar mínimamente el coste de las consultas sobre las estructuras.

[1] Ver anexo A para una introducción a las 2-estructuras.
[2] El manual está disponible en el anexo E.
[3] Se han omitido los atributos y las operaciones de las clases por claridad.

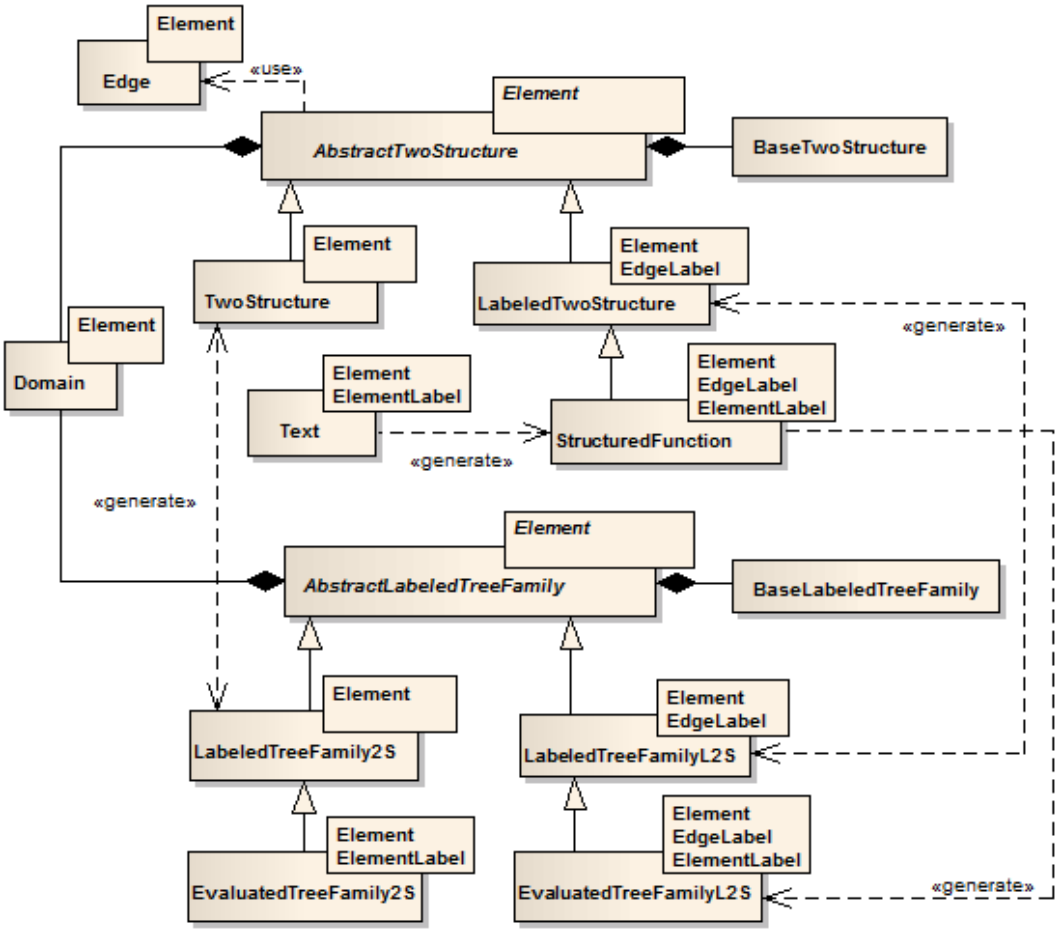

Ilustración 2.1 Diagrama de clases con el modelado y el diseño de las 2-estructuras.

Las 2-estructuras son básicamente conjuntos disjuntos de aristas y por ello se han implementado usando una conocida representación eficiente en árbol [16] que tiene un coste amortizado asíntoticamente óptimo. Las *labeled tree families* se han implementado trasladando de forma natural su definición a estructuras de datos típicas con las optimizaciones que su semántica permitía.

2.2. Controles gráficos para la visualización

En la ilustración 2.2 se muestra el diseño del visualizador de las 2-estructuras y el de las labeled tree families. El primero tiene desacoplada toda la lógica del dibujado y la interacción, que se encuentra contenida en la clase `TwoStructureCanvas`, del control gráfico propiamente dicho (`TwoStructureViewer`). De esta forma, el visualizador de las labeled tree families (`LabeledTreeFamilyViewer`) puede reutilizar los métodos

de dibujado para mostrar las 2-estructuras que etiquetan cada nodo interno del árbol.

Además, los algoritmos de posicionamiento de los nodos para cada estructura se encuentran en clases separadas de las de los controles. En concreto en `TwoStructureLayout` y en `GeneralTreeLayout`.

En el caso de las 2-estructuras el objetivo es dibujar un grafo completo. Por ello se opta por un posicionamiento de los nodos en forma de polígono regular. Se tratan como situaciones especiales las 2-estructuras, con pocos elementos, que etiquetan los nodos de las labeled tree families. En dichas situaciones los nodos se posicionan de forma que faciliten las conexiones con los hijos. Para las labeled tree families se implementa el algoritmo de Walker [14] que sirve para situar los nodos de un árbol con un número arbitrario de hijos en cada nivel.

Ambos visualizadores se han implementado como controles gráficos (Swing) de Java con `JPanel` como superclase de partida.

Debido a que el número de elementos de las 2-estructuras que etiquetan los nodos de las labeled tree families puede ser arbitrariamente grande, sólo se dibujan en el interior de los nodos las 2-estructuras con como máximo tres elementos. Para poder ver una labeled tree family junto a la 2-estructura con cuatro o más elementos que etiqueta un nodo, se ha diseñado un control gráfico de doble panel (`LabeledTreeFamilySplitViewer`) tal que el panel izquierdo muestra la labeled tree family y el derecho la 2-estructura del nodo que se seleccione. De nuevo este último componente es un control gráfico de Java que hereda esta vez de `JSplitPanel`.

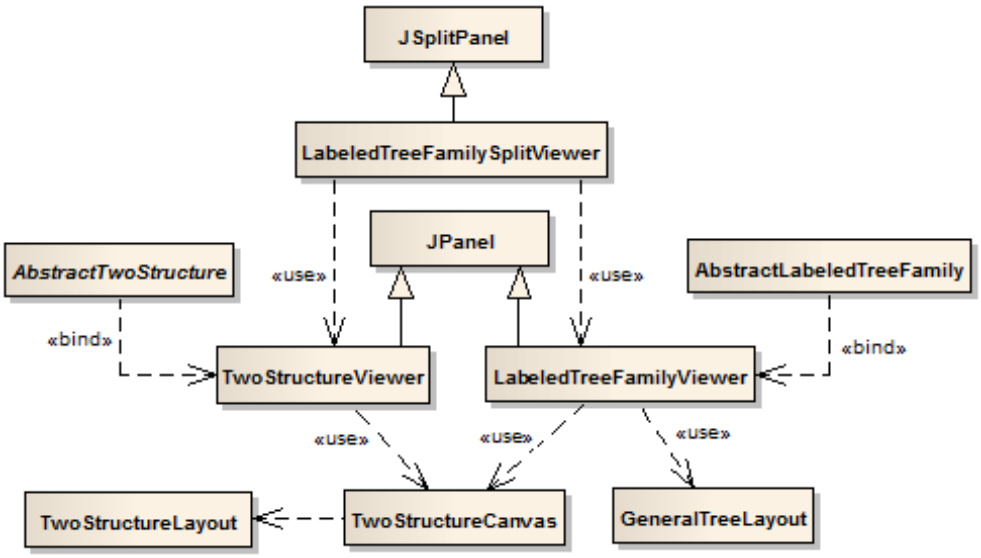

Ilustración 2.2 Controles gráficos de visualización de 2-estructuras y sus formas.

2.3. Arquitectura y diseño del programa interactivo

En la ilustración 2.3 se muestra la división en tres capas de la aplicación de escritorio que permite analizar, transformar y visualizar las distintas estructuras del dominio del problema.

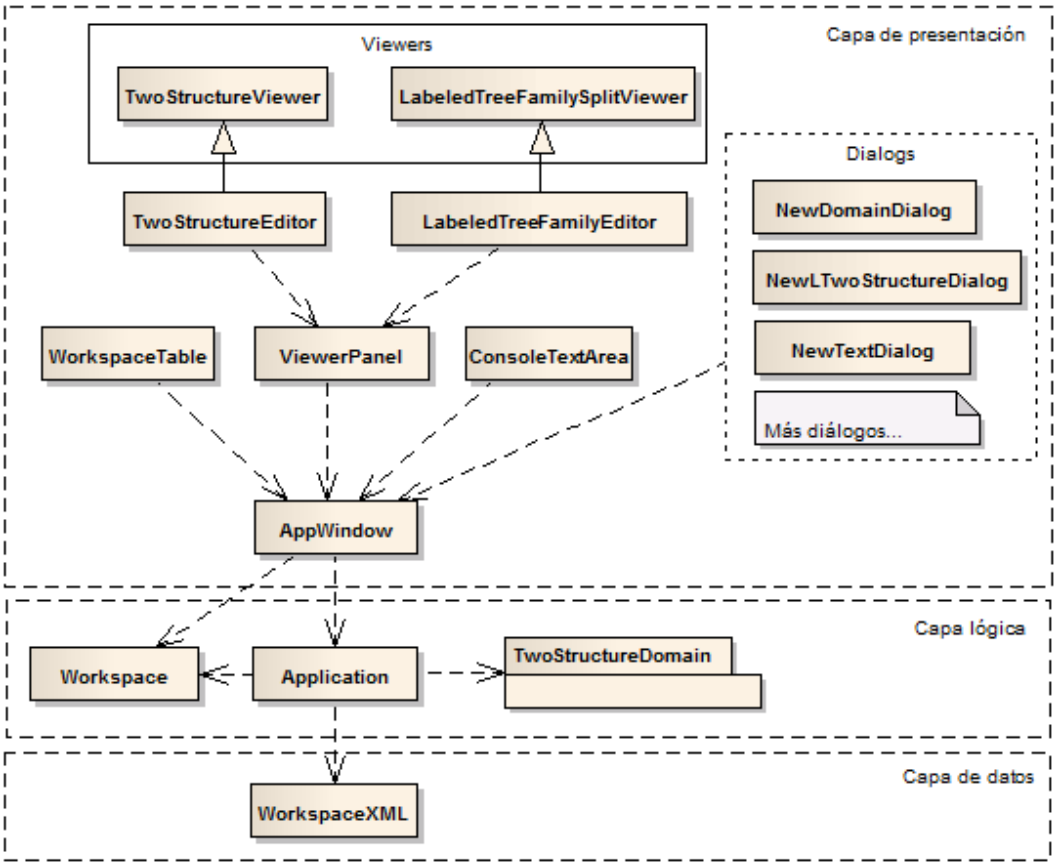

Ilustración 2.3 Arquitectura en tres capas del programa interactivo.

En la capa lógica está la clase `Application` que se encarga de construir la interfaz y procesar las respuestas a los distintos eventos. También está la clase `Workspace` que mantiene un diccionario de nombres de variables asociados a instancias de las estructuras creadas en una sesión. La idea del espacio de trabajo es la misma que la que utiliza el conocido software matemático MATLAB.

El espacio de trabajo de una sesión puede ser almacenado de manera permanente en un fichero XML utilizando la clase `WorkspaceXML` de la capa de datos.

La ventana principal del programa se divide en tres áreas que se corresponden con las clases `WorkspaceTable` (la tabla en donde se lista las variables de la sesión),

`ConsoleTextArea` (un cuadro de texto donde se lista información sobre las estructuras seleccionadas) y `ViewerPanel` (el panel donde se muestra las representaciones gráficas de las estructuras).

El `ViewerPanel` contiene una especialización de los visores descritos en la sección anterior a los que se les ha añadido opciones de edición y transformación. Por ejemplo, permite seleccionar varias aristas de una 2-estructura y hacerlas equivalentes o seleccionar varios elementos de una 2-estructura y generar la subestructura correspondiente.

Para mantener una separación estanca entre las distintas capas de forma que sólo las capas superiores dependan de los servicios ofrecidos por las capas inferiores, se implementa el patrón *Observador*. De esta manera la interfaz, ante las peticiones del usuario, llama a los métodos de la capa lógica donde se realizan las modificaciones del modelo (como el espacio de trabajo) y los controles de la interfaz, suscritos a los eventos de las partes afectadas por los cambios, son notificados sincronizando su estado con el del modelo. El diseño se corresponde con lo que algunos autores llaman una arquitectura Modelo-Vista-Controlador.

.

3 Asociación de un árbol derivado a las cadenas generadas por una gramática HG

Las gramáticas de núcleo[4] no proporcionan un árbol derivado a las cadenas del lenguaje [11,12]. En este capítulo se propone una extensión de las gramáticas de núcleo que asocia explícitamente un árbol derivado a las cadenas generadas usando las mismas operaciones de concatenación y de wrapping empleadas en el formalismo. Para ello se hace uso de los conceptos sobre *textos alternantes* de la teoría de las 2-estructuras[5].

Esto permite dotar de una estructura a las palabras generadas para poder compararlas con las dadas por otras gramáticas que generan la misma clase de lenguajes[8], como las gramáticas de adjunción de árboles. El interés de este trabajo, y su diferencia con el de otros trabajos previos como [12], radica en que se utiliza un formalismo robusto (las 2-estructuras) para expresar la estructura de una palabra de una forma 'natural' (basándose en las operaciones de concatenación y de wrapping originales).

3.1. Extensión de las gramáticas de núcleo

En la nueva gramática propuesta vamos a realizar derivaciones de *textos con núcleo*, que definiremos como 3-tuplas pertenecientes a:

$$HT = \langle (C \to V_T) \times (C^+ \times C^+) \times (C^+ \times C^+) \rangle$$

donde $C^+ = \cup_{k \geq 1} C^k$ y, C y V_T son dos conjuntos arbitrarios. Notar que al trabajar con textos con núcleo trabajaremos con pares ordenados de secuencias y no con pares de ordenes lineales (revisar la definición de un texto en A.2), ya que desde el punto de vista gramatical nos interesará manipular secuencias. Como veremos adelante, del conjunto C sólo necesitaremos que se pueda generar un elemento distinto a los de una secuencia ya generada a partir de otro elemento cualquiera del conjunto, es decir, que tenga una función sucesor definida. Por comodidad consideramos que C es siempre el conjunto de los números naturales $\mathbb{N}$ que cumple dicha condición.

Adelantamos que durante la generación del árbol derivado, las operaciones de concatenación y wrapping concatenarán subárboles asociados a textos con distinta interpretación de la dirección. Para hacerlo de manera consistente se necesitará saber de dicha interpretación en las posiciones (nodos del árbol correspondiente) donde se apliquen y, por tanto, en vez de trabajar con textos con núcleo 'puros' se empleará una extensión de los mismos. Como ya se verá, sólo hay dos puntos posibles de concatenación, en la raíz o en un nodo 'pie', por lo que en vez de usar HT durante la derivación, en realidad se usará el conjunto $HT^+ = HT \times V_A^2$ donde

[4] Ver anexo B.4 para la definición de las gramáticas de núcleo.
[5] Ver anexo A para una introducción a las 2-estructuras y los textos alternantes.

$V_A = \{\rightarrow, \leftarrow\}$. El elemento $\rightarrow$ indicará que la interpretación de la dirección es de izquierda a derecha mientras que el elemento $\leftarrow$ indicará la otra dirección. Usaremos el operador $\neg$ sobre V_A con su significado 'habitual' de opuesto. Es decir, $\neg \rightarrow = \leftarrow$ y $\neg \leftarrow = \rightarrow$.

Dado un texto con núcleo extendido $\tau = \langle \lambda, (u{\uparrow}v), (p{\uparrow}q), (r, f)\rangle$, diremos que el texto asociado a τ es $(\lambda, \rho_1, \rho_2)$ donde ρ_1 y ρ_2 son los ordenes lineales inducidos por las secuencias uv y pq respectivamente. Por otra parte, diremos que el árbol asociado a τ es el árbol ordenado inducido por su texto asociado y que la interpretación de la dirección en la raíz es r. Para hablar de una interpretación de la dirección de τ en concreto, usaremos $dir_+(\tau)$ para denotar a r y $dir_-(\tau)$ para denotar a f. Definiremos el *reverso o inverso de* τ como $\langle \lambda, \rho_1, (rev(q), rev(p)), (\neg r, \neg f)\rangle$ y lo denotaremos con $rev(\tau)$. Diremos que la longitud de τ, $len(\tau)$, es $|u| + |v|$ con el significado habitual de la operación al utilizar secuencias.

Ejemplo 3.1. Sea $\tau = \langle \lambda, \rho_1, \rho_2, \theta\rangle \in HT^+$ con $\lambda = \{(1, a), (2, a), (3, b), (4, b)\}$, $\rho_1 = (1, 2{\uparrow}3, 4)$, $\rho_2 = (1, 4{\uparrow}2, 3)$ y $\theta = (\rightarrow, \rightarrow)$. La longitud de τ es $len(\tau) = |(1, 2)| + |(3, 4)| = 2 + 2 = 4$. Y el reverso de τ es $\mu = \langle \lambda, \rho_1, (3, 2{\uparrow}1, 4), (\leftarrow, \leftarrow)\rangle$.

Respecto a una gramática de núcleo en su concepción original, la versión extendida modifica las operaciones de concatenación y de wrapping, y la relación de derivación $\Rightarrow$. Por tanto, sólo se van a redefinir éstas enteniéndose que el resto de la definición de la gramática sigue siendo aplicable. Para que esto sea así, se empleará en la relación de derivación la siguiente función de conversión de una cadena con núcleo a un texto con núcleo extendido usando un natural s como 'semilla generadora'.

$ht : (V_T^* \times V_T^*) \times \mathbb{N} \rightarrow HT^+$:

$$ht(u_1 \dots u_n{\uparrow}v_1 \dots v_m, s) = (\lambda, \rho_1, \rho_2, \theta)$$

donde:

$$\begin{aligned}
\lambda &= \{(s, u_1), \dots, (s + n - 1, u_n), (s + n, v_1), \dots, (s + n + m - 1, v_m)\} \\
\rho_1 &= \langle s, \dots, s + n - 1{\uparrow}s + n, \dots, s + n + m - 1\rangle \\
\rho_2 &= \langle s + n + m - 1, \dots, s + n{\uparrow}s + n - 1, \dots, s\rangle = rev(\rho_1) \\
\theta &= (\leftarrow, \leftarrow)
\end{aligned}$$

$C_{i,n} : (HT^+)^n \rightarrow HT^+$ es la nueva operación de concatenación:

$$C_{i,n}(\tau_1, \dots, \tau_i, \dots, \tau_n) = \langle \cup_{t=1}^{n} \lambda_t, u_1 v_1 \dots u_i{\uparrow}v_i \dots u_n v_n, p_1 q_1 \dots p_i{\uparrow}q_i \dots p_n q_n, (\rightarrow, f_i)\rangle$$

donde $\tau_j = \langle \lambda_j, u_j{\uparrow}v_j, p_j{\uparrow}q_j, (r_j, f_j)\rangle$ con $(1 \leq j \leq n)$.

$W : (HT^+)^2 \rightarrow (HT^+)$ es la nueva operación de wrapping:

$$W(\tau_1, \tau_2) = \langle \lambda_1 \cup \lambda_2, u_1 u_2 \uparrow v_2 v_1, p_1 p_2 \uparrow q_2 q_1, (r_1, f_2) \rangle$$

donde $\tau_j = \langle \lambda_j, u_j \uparrow v_j, p_j \uparrow q_j, (r_j, f_j) \rangle$ con $(1 \leq j \leq 2)$.

Antes de poder redefinir la relación de derivación tenemos que definir dos funciones auxiliares, $left$ y $comp$. La primera es utilizada para asegurarse de que si un texto con núcleo extendido tiene $\rightarrow$ como dir_+, éste sea invertido. La segunda función se emplea para asegurarse de que si un texto con núcleo extendido no tiene como dir_+ la opuesta a una dada, éste sea invertido.

$left : HT^+ \rightarrow HT^+$:

$$left(\tau) = \begin{cases} rev(\tau), & \text{si } dir_+(\tau) = \rightarrow \\ \tau, & \text{si } dir_+(\tau) = \leftarrow \end{cases}$$

$comp : HT^+ \times V_A \rightarrow HT^+$:

$$comp(\tau, a) = \begin{cases} rev(\tau), & \text{si } dir_+(\tau) = a \\ \tau, & \text{si } dir_+(\tau) \neq a \end{cases}$$

La relación de derivación $\Rightarrow$ se define como sigue:

- $\tau \overset{0}{\Rightarrow} \tau$ para todo $\tau \in HT^+$.

- Si $A \rightarrow W(\sigma_1, \sigma_2) \in P$ entonces $(A, s) \overset{k}{\Rightarrow} W(\tau_1, comp(\tau_2, dir_-(\tau_1)))$ donde:

  $$\left((\sigma_j, next(s, j)) \overset{k_j}{\Longrightarrow} \tau_j, \text{ si } \sigma_j \in V_N \right), \left(ht(\sigma_j, next(s, j)) \overset{k_j}{\Longrightarrow} \tau_j, \text{ si } \sigma_j \in V_T^* \times V_T^* \right)$$

  con $(1 \leq j \leq 2)$, $k = 1 + \sum_{1 \leq j \leq 2} k_j$ y $next(s, j) = \begin{cases} s, & \text{si j=1} \\ s + \sum_{t=1}^{j-1} len(\tau_t), & \text{si j>1} \end{cases}$

- Si $A \rightarrow C_{i,n}(\sigma_1, \ldots, \sigma_n) \in P$ entonces $(A, s) \overset{k}{\Rightarrow} C_{i,n}(left(\tau_1), \ldots, left(\tau_n)))$ donde:

  $$\left((\sigma_j, next(s, j)) \overset{k_j}{\Longrightarrow} \tau_j, \text{ si } \sigma_j \in V_N \right), \left(ht(\sigma_j, next(s, j)) \overset{k_j}{\Longrightarrow} \tau_j, \text{ si } \sigma_j \in V_T^* \times V_T^* \right)$$

  con $(1 \leq j \leq n)$, $k = 1 + \sum_{1 \leq j \leq n} k_j$ y $next(s, j) = \begin{cases} s, & \text{si j=1} \\ s + \sum_{t=1}^{j-1} len(\tau_t), & \text{si j>1} \end{cases}$

Destacar que de la manera en la que hemos definido la relación de derivación, ésta nos asegura que todo texto asociado a un texto con núcleo extendido resultado de una derivación sea alternante.

El lenguaje generado por una gramática de núcleo extendida queda definido por todos los $(\lambda(u_1)\cdot\ldots\cdot\lambda(u_n)) \in V_T^*$ tales que $(S,1) \overset{*}{\Rightarrow} (\lambda,(u_1,\ldots,u_i\uparrow u_i,\ldots,u_n),\rho_2,\theta)$.

Dada una palabra $(\lambda(u_1)\cdot\ldots\cdot\lambda(u_n)) \in V_T^*$ tal que $(S,1) \overset{*}{\Rightarrow} \tau$ donde $\tau = (\lambda,(u_1,\ldots,u_i\uparrow u_i,\ldots,u_n),\rho_2,\theta)$, su árbol derivado asociado viene determinado por el árbol asociado a τ.

3.2. Ejemplo ilustrativo

La gramática de núcleo *extendida*[6] $G = (\{S,T\},\{a,b,c,d\},S,P)$ genera el lenguaje $\{a^nb^nc^nd^n \mid n \geq 0\}$ donde P es como sigue.

$$P = \{S \to C_{1,1}(\epsilon\uparrow\epsilon),\quad S \to C_{2,3}(a\uparrow\epsilon,T,d\uparrow\epsilon),\quad T \to W(S,b\uparrow c)\}$$

La derivación de la cadena *aabbccdd* junto a la de su árbol derivado asociado se descompone en los siguientes dos grupos de pasos. Vamos a presentarlos en dos grupos separados para verlo más claramente.

En el primer grupo detallamos la reglas de reescritura aplicadas.

$$(S,1) \overset{5}{\Rightarrow} \overbrace{C_{2,3}(\underbrace{left(ht(a\uparrow\epsilon,1))}_{\tau_9},\underbrace{left((T,3))}_{\tau_{10}},\underbrace{left(ht(d\uparrow\epsilon,13))}_{\tau_{11}})}^{\tau_{12}}$$

que depende de $(T,3) \overset{4}{\Rightarrow} W(\underbrace{(S,3)}_{\tau_7},\underbrace{comp(ht(b\uparrow c,11),dir_-(\tau_7))}_{\tau_8})$

que depende de $(S,3) \overset{3}{\Rightarrow} C_{2,3}(\underbrace{left(ht(a\uparrow\epsilon,3))}_{\tau_4},\underbrace{left((T,5))}_{\tau_5},\underbrace{left(ht(d\uparrow\epsilon,9))}_{\tau_6})$

que depende de $(T,5) \overset{2}{\Rightarrow} W(\underbrace{(S,5)}_{\tau_2},\underbrace{comp(ht(b\uparrow c,7),dir_-(\tau_2))}_{\tau_3})$

que depende de $(S,5) \overset{1}{\Rightarrow} C_{1,1}(\underbrace{left(ht(\epsilon\uparrow\epsilon,5))}_{\tau_1})$

En el segundo grupo listamos los resultados de aplicar las operaciones de conversión, de concatenación y de wrapping para obtener el texto con núcleo extendido final.

[6] En el anexo B.4 se encuentra el mismo ejemplo para gramáticas de núcleo. Su 'aparencia' es idéntica puesto que lo que cambia son las definiciones de la relación de derivación y las operaciones de concatenación y de wrapping.

$$\tau_1 = \langle\{(5,\epsilon),(6,\epsilon)\},(5{\uparrow}6),(6{\uparrow}5),(\leftarrow,\leftarrow)\rangle$$
$$\tau_2 = C_{1,1}(\tau_1) = \langle\lambda_1,(5{\uparrow}6),(6{\uparrow}5),(\leftarrow,\leftarrow)\rangle$$
$$\tau_3 = \langle\{(7,b),(8,c)\},(7{\uparrow}8),(7{\uparrow}8),(\rightarrow,\rightarrow)\rangle$$
$$\tau_4 = \langle\{(3,a),(4,\epsilon)\},(3{\uparrow}4),(4{\uparrow}3),(\leftarrow,\leftarrow)\rangle$$
$$\tau_5 = left(W(\tau_2,\tau_3)) = \langle\lambda_2\cup\lambda_3,(5,7{\uparrow}8,6),(6,7{\uparrow}8,5),(\leftarrow,\rightarrow)\rangle$$
$$\tau_6 = \langle\{(9,d),(10,\epsilon)\},(9{\uparrow}10),(10{\uparrow}9),(\leftarrow,\leftarrow)\rangle$$
$$\begin{aligned}\tau_7 &= C_{2,3}(\tau_4,\tau_5,\tau_6)\\ &= \langle\lambda_4\cup\lambda_5\cup\lambda_6,(3,4,5,7{\uparrow}8,6,9,10),(4,3,6,7{\uparrow}8,5,10,9),(\rightarrow,\rightarrow)\rangle\end{aligned}$$
$$\tau_8 = \langle\{(11,b),(12,c)\},(11{\uparrow}12),(12{\uparrow}11),(\leftarrow,\leftarrow)\rangle$$
$$\tau_9 = \langle\{(1,a),(2,\epsilon)\},(1{\uparrow}2),(2{\uparrow}1),(\leftarrow,\leftarrow)\rangle$$
$$\begin{aligned}\tau_{10} &= left(W(\tau_7,\tau_8))\\ &= left(\langle\lambda_7\cup\lambda_8,(3,4,5,7,11{\uparrow}12,8,6,9,10),(4,3,6,7,12{\uparrow}11,8,5,10,9),(\rightarrow,\leftarrow)\rangle)\\ &= \langle\lambda_7\cup\lambda_8,(3,4,5,7,11{\uparrow}12,8,6,9,10),(9,10,5,8,11{\uparrow}12,7,6,3,4),(\leftarrow,\rightarrow)\rangle\end{aligned}$$
$$\tau_{11} = \langle\{(13,d),(14,\epsilon)\},(13{\uparrow}14),(14{\uparrow}13),(\leftarrow,\leftarrow)\rangle$$
$$\begin{aligned}\tau_{12} &= C_{2,3}(\tau_9,\tau_{10},\tau_{11})\\ &= \langle\lambda_9\cup\lambda_{10}\cup\lambda_{11},(1,2,3,4,5,7,11{\uparrow}12,8,6,9,10,13,14),\\ &\quad (2,1,9,10,5,8,11{\uparrow}12,7,6,3,4,14,13),(\rightarrow,\rightarrow)\rangle\end{aligned}$$

A partir de $\tau_{12} = (\lambda,(u{\uparrow}v),\rho,\theta)$, se puede obtener la palabra derivada aplicando λ a cada elemento de la secuencia uv, $(\lambda(1)\cdot\ldots\cdot\lambda(14)) = aabbccdd$, y se puede obtener el texto $(\lambda,(1,2,3,4,5,7,11,12,8,6,9,10,13,14),(2,1,9,10,5,8,11,12,7,6,3,4,14,13))$ que representa el árbol derivado de la ilustración 3.1[7].

Para entender mejor cómo va quedando definido el árbol derivado durante el proceso de derivación, vamos a analizar un paso de la derivación en la que aparezca una operación de wrapping, por ejemplo al calcular τ_5 a partir de τ_2 y τ_3.

En la ilustración 3.2 tenemos los árboles derivados asociados a τ_2, τ_3 y τ_5 respectivamente. Podemos ver que el árbol de τ_5 es el resultado de concatenar el árbol de τ_3 al árbol de τ_2 por el punto donde la segunda secuencia de τ_2 está dividida.

Notar que las interpretaciones de las direcciones de los nodos involucrados son complementarias porque τ_3 es el resultado de aplicar la función *comp* a $ht(b{\uparrow}c,7)$

[7] Realmente es la representación del árbol derivado junto a la información del texto a partir de la cual se genera.

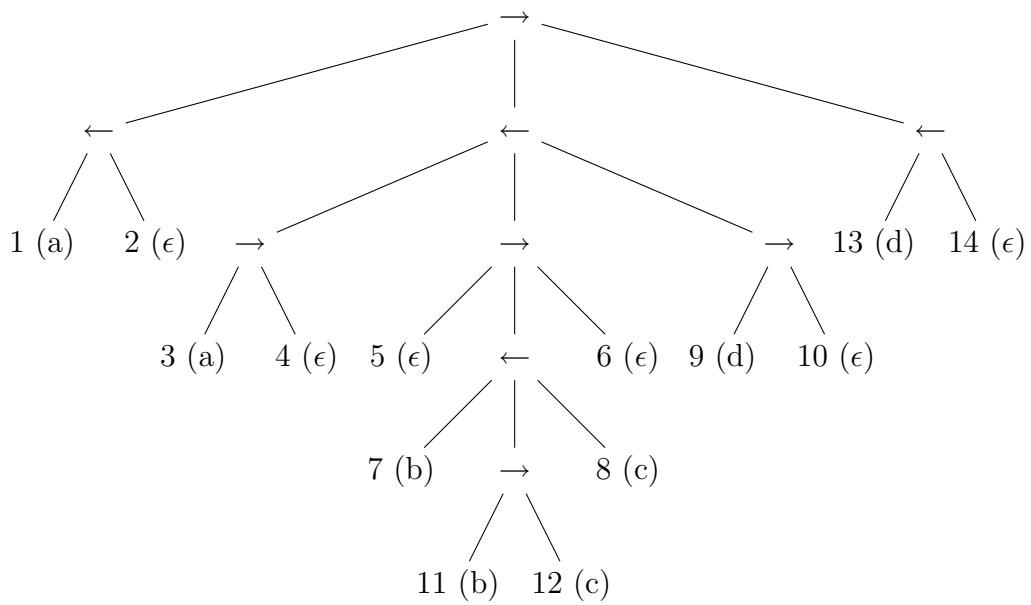

Ilustración 3.1 Árbol derivado asociado a la palabra *aabbccdd*.

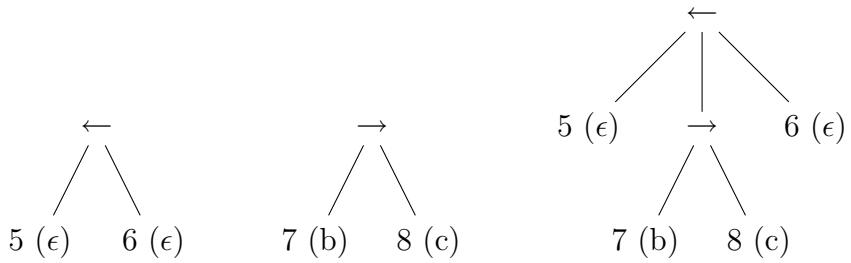

Ilustración 3.2 Árbol derivados asociados a τ_2, τ_3 y τ_5.

con $dir\_(\tau_2)$. En caso contrario ambos nodos tendrían la misma interpretación de la dirección y el texto resultante no sería alternante.

Ahora veamos un paso de la derivación en la que aparezca una operación de concatenación, por ejemplo al calcular τ_7 a partir de τ_4, τ_5 y τ_6.

En la ilustración 3.3 tenemos los árboles derivados asociados a τ_4, τ_6 y τ_7 respectivamente. Podemos observar que el nuevo árbol formado, el asociado a τ_7, es el resultado de crear un nuevo nodo raíz cuyos hijos son los árboles asociados a τ_4, τ_5 y τ_6 (los argumentos de la operación de concatenación). Para que esto sea posible, todos los nodos hijos tienen que tener una interpretación de la dirección distinta al nodo raíz. Como la interpretación para el nodo raíz se fija en →, se debe 'obligar' con la función $left$ que los hijos tengan la contraria.

Para este ejemplo no hace falta invertir ningún texto con núcleo extendido pero, por ejemplo, al calcular τ_{12} la función $left$ tiene efecto en τ_{10}.

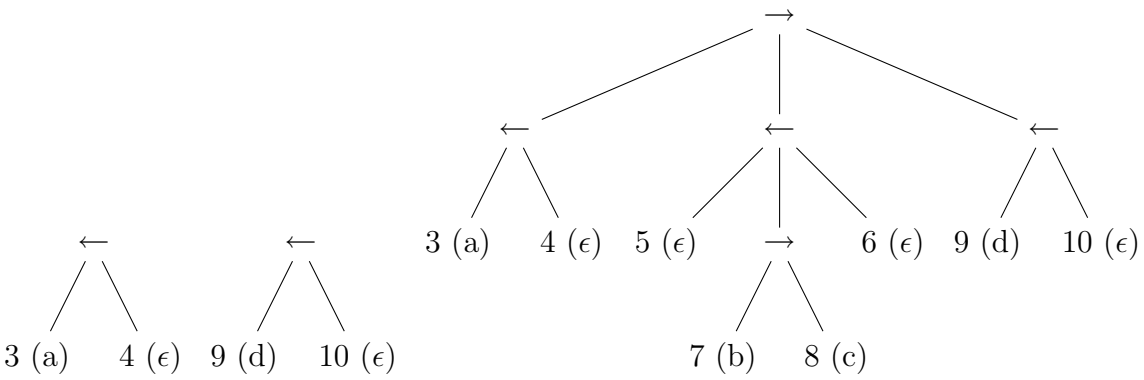

Ilustración 3.3 Árbol derivados asociados a τ_4, τ_6 y τ_7.

.

4 Generación de una gramática TAG a partir de una frase con dependencias

En este capítulo se presenta un algoritmo capaz de construir a partir de una cadena con dependencias entre sus símbolos[8] una gramática TAG[9] que la genere (si hay alguna) y un árbol derivado obtenido al generarla. Para ello se describe previamente una estructura llamada árbol de dependencias que es usada por el método.

En el anexo D se puede ver una implementación del algoritmo, el cual ha sido incluido e integrado dentro del paquete de software desarrollado para las 2-estructuras. De esta forma se puede generar y visualizar el árbol de dependencias (y algunas estructuras de descomposición intermedias usadas para generarlo) con el programa interactivo.

4.1. Definición de un árbol de dependencias

Consideremos la gramática $G = (\{a, b, c, d\}, \{S, T\}, \{\alpha_1\}, \{\beta_1\}, S)$ representada en la ilustración 4.1 que genera el lenguaje $\{a^n b^n c^n d^n \mid n \geq 1\}$.

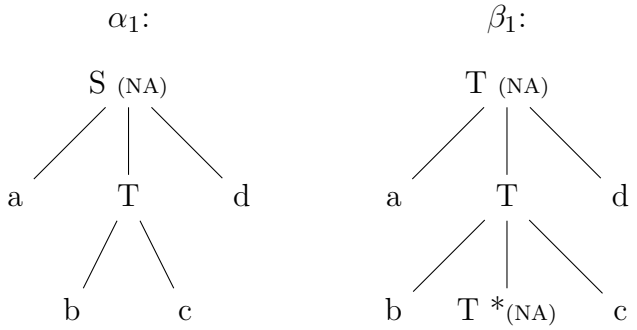

Ilustración 4.1 Gramática de adjunción de árboles que genera el lenguaje $a^n b^n c^n d^n$

El árbol derivado de la palabra *aabbccdd* generado por la gramática G está dibujado en la parte izquierda de la ilustración 4.2. La representación gráfica de la palabra junto a sus depedencias se puede encontrar en la ilustración 4.3.

Uno puede notar observando el árbol derivado que es capaz de asociar a cada nodo intermedio una etiqueta, sobre un conjunto finito cualquiera, de forma que ésta coincida para nodos intermedios que tienen como hijos directos a terminales que se generaron a la vez a partir del mismo árbol elemental y es distinta en cualquier otro caso. Al árbol resultado de sustituir cada no-terminal del árbol derivado por su

[8] Nos referimos por cadena a la palabra de un lenguaje formal, que en un idioma se correspondería con un frase (ver anexo B.2 para una explicación más detallada y una descripción de las dependencias del lenguaje natural).

[9] Ver anexo B.3 para la definición de las TAG.

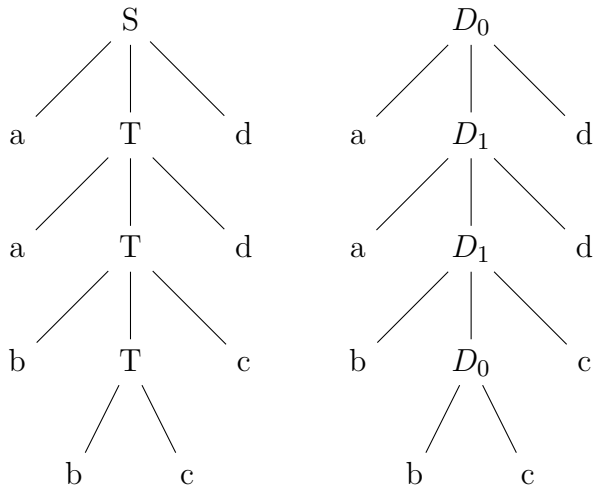

Ilustración 4.2 Árbol derivado y árbol de dependencias para la palabra *aabbccdd*.

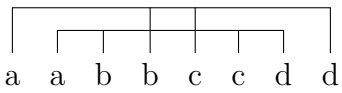

Ilustración 4.3 La palabra *aabbccdd* junto a sus dependencias.

etiqueta le llamaremos *árbol de dependencias* y a los símbolos usados para etiquetar le llamaremos *dependencias*. Para el ejemplo visto anteriormente, dicho árbol está dibujado a la derecha de la ilustración 4.2.

Por otra parte, se puede ver que dado un árbol de dependencias uno puede construir una grámatica TAG que sea capaz de generarlo. Para una explicación de dicha correspondencia se remite al anexo C.

Por tanto, el problema a resolver se puede replantear como encontrar un árbol de dependencias que represente la estructura de un árbol derivado de una palabra generable por una gramática TAG respetando las dependencias dadas.

4.2. División del problema en dos casos

Dada una palabra $w = c_1...c_n$ con m dependencias (con $m \leq n$), usaremos los elementos del subconjunto $D = \{D_0, .., D_m\}$ para etiquetar los n carácteres según las dependencias definidas entre ellos. Por tanto, una entrada al problema planteado es un par formado por una secuencia de símbolos del alfabeto y una secuencia de elementos (dependencias) de D.

A la hora de abordar la generación de un árbol de dependencias podemos distinguir entre árboles con una sola espina, es decir, todo nodo intermedio tiene como máximo un nodo no hoja como hijo directo, y árboles con varias espinas o ramas independientes. A los del primer tipo les llamaremos árboles de dependencias *simples* y a los del segundo árboles de dependencias *complejos*.

Desde el punto de vista de una palabra y sus dependencias (una entrada), los dos clases de árboles quedan caracterizados por las siguientes dos propiedades.

- Un árbol de dependencias es complejo si y sólo si al dibujar gráficamente las dependencias entre los símbolos de la palabra existe un par de líneas de dos dependencias distintas que ni se cruzan ni una anida a la otra.

- Sea $seq(D_i)$ la secuencia de posiciones de la palabra etiquetadas con D_i. El mínimo y el máximo valor de cada secuencia $seq(D_i)$ define un intervalo $int(D_i)$. Un árbol de dependencias es complejo si y sólo si existen dos intervalos $int(D_i)$ y $int(D_j)$ con $i \neq j$ que son disjuntos entre sí.

La construcción de un árbol de dependencias complejo se puede descomponer en la generación por separado de los subárboles asociados a cada espina del árbol y la combinación de los mismos. Por ello, primero se explicará cómo resolver el problema para un árbol de dependencias simple y luego para uno complejo (el caso general).

4.3. Método para generar un árbol de dependencias simple

Sea un árbol de dependencias simple tal que cada nodo interno tiene exactamente un hijo hoja en cada uno de los dos lados en la que la única espina existente divide al árbol. Entonces, la secuencia de dependencias que etiqueta cada terminal de la palabra asociada al árbol se puede dividir en dos subsecuencias i y d tales que i es la secuencia inversa de d. Esto se cumple porque cada subsecuencia se corresponde con el recorrido desde la raíz al nodo en el nivel más bajo del lateral izquierdo y su recorrido inverso por el lateral derecho.

Lo anterior se puede extender al caso en que un nodo interno tiene más de un hijo hoja asociado a un lado si se trata a todos ellos como si fueran uno solo. Este hecho se corresponde a la existencia de dos dependencias iguales contiguas en la secuencia de dependencias.

Como última posibilidad, si un nodo interno no tiene al menos un hijo en un lateral se puede considerar como si dicho nodo si lo tuviera y fuera ϵ (la palabra vacía).

Para ver la primera afirmación, recuperemos el ejemplo mostrado en la ilustración 4.3. La palabra junto a sus dependencias forman la entrada ($aabbccdd$, $D_0D_1D_1D_0D_0$

$D_1D_1D_0$). Si la caracterizamos, tal como se detalló en la sección anterior, podemos deducir que la solución buscada tiene que ser un árbol de dependencias simple.

Por otra parte, se puede observar que es posible dividir las dependencias en dos subsecuencias iguales ($D_0D_1D_1D_0$) que, trivialmente, cumplen la propiedad de ser una la inversa de la otra.

Si nos fijamos en el árbol derivado de la palabra *aabbccdd*, mostrado en la parte izquierda de la ilustración 4.2, se puede ver que cada subsecuencia en la que se puede dividir las dependencias se corresponde con los terminales presentes en cada lateral del árbol.

Por tanto, un algoritmo para hallar un árbol de dependencias simple consistiría en conseguir partir la secuencia de dependencias en dos subsecuencias tales que insertando, si es necesario, dependencias ficticias (asociadas a ϵ) una sea la inversa de la otra.

No obstante, hay que tener en cuenta que no todos las inserciones llevan a secuencias de dependencias compatibles con un árbol de dependencias válido. Una secuencia será compatible si y sólo si ninguna dependencia situada entre dos dependencias iguales está también antes o después de las dependencias que la rodean. Esto tiene que ser así porque la operación de adjunción o bien se comporta como una operación de concatenación por delante o detrás de un grupo de nodos de dependencias o bien divide en dos a un grupo de nodos con la misma dependencia quedando situado sobre las dos partes resultantes.

También es necesario ver que puede haber más de una manera válida de insertar dependencias ficticias en la secuencia, lo que sugiere, unido al anterior hecho, que se debe realizar una búsqueda en el espacio de soluciones. Además en esta búsqueda interesará que el número de dependencias introducidas sea mínimo, aunque esta condición es prescindible.

En el anexo D.1 se puede ver una implementación del algoritmo.

4.4. Método para generar un árbol de dependencias complejo

La idea que se plantea para resolver estos árboles consiste en identificar cada rama independiente de tal manera que se agrupen todas las dependencias pertenencientes a dicha rama bajo una dependencia que hará de representante y será hijo en un lateral de una rama superior. Es decir, la rama independiente equivaldrá a un nodo hoja de una espina superior. Después se resolverá cada rama independiente, con el algoritmo ya visto, que dará lugar a un subárbol que será sustituido por su nodo representante.

Para ayudar a entender la explicación se va a presentar un ejemplo. Supongamos que se tiene la siguiente entrada: $((eabbbtptcddvpvdere), (D_0D_1D_2D_3D_4D_5 D_6D_5D_1D_2D_3D_7D_6D_7D_4D_0D_8D_0))$. Su representación gráfica está disponible en la ilustración 4.4.

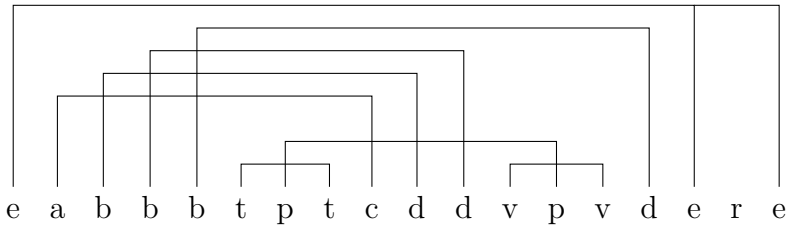

Ilustración 4.4 Palabra con dependencias disjuntas.

Claramente se aprecia que las dependencias D_5, D_7 y D_8, asociados a los terminales t, v y r respectivamente, son disjuntas entre sí. Para eliminar dicho problema se propone agrupar las dependencias D_5 bajo una dependencia D_6, las D_7 bajo otra dependencia D_6 y la D_8 bajo una dependencia D_0. La ilustración 4.5 lo muestra gráficamente. Para el caso de D_8 y D_0 se puede apreciar una agrupación adicional. Ésta es debida a que, como se explicó para el algoritmo de la sección anterior, si hay varias dependencias iguales contiguas se consideran como si fuera una sola.

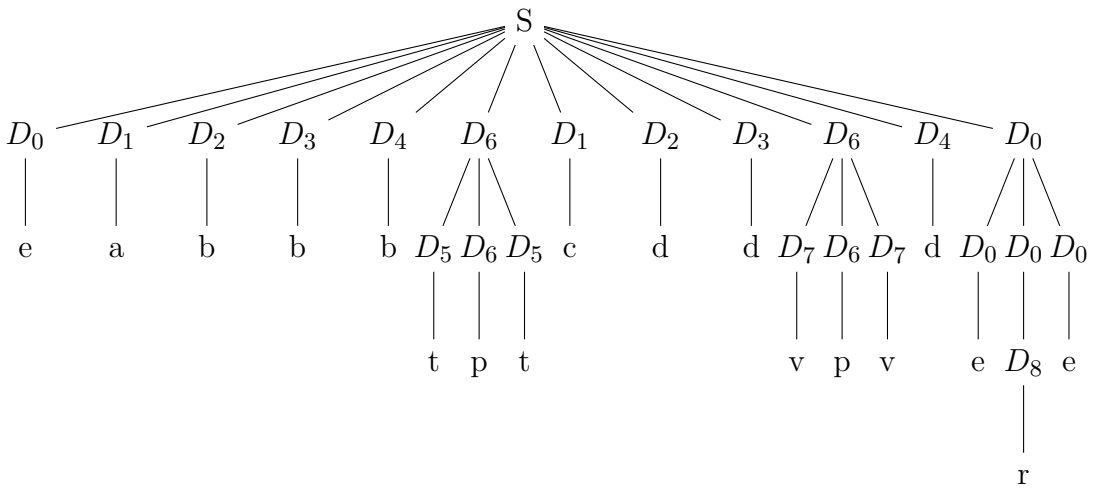

Ilustración 4.5 Subdivisión de una entrada en ramas independientes.

Una vez realizadas las agrupaciones, cada nivel del árbol se puede procesar como una entrada del algoritmo para árboles de dependencias simples y combinar los árboles resultantes para construrir el árbol de dependencias final.

La siguiente cuestión a plantear es cómo se puede llegar a identificar correctamente el grupo de dependencias que forma una rama independiente y averiguar

cuál es la dependencia que la representa. Como solución a este problema se propone un método iterativo/recursivo que dada una secuencia de dependencias genera una nueva secuencia formada como resultado de sustituir algunas subsecuencias de dependencias contiguas por otra dependencia en la secuencia de entrada. El algoritmo termina cuando la secuencia de entrada está constituida por una sola dependencia.

Además de todas las dependencias involucradas en la secuencia original, también puede ser necesario disponer de una superdependencia, que denotaremos por S, que será padre de todas aquellas ramas independientes que no tengan en común ninguna dependencia.

A continuación se describen los pasos más importantes del algoritmo:

- Se lee toda la secuencia de dependencias y se calcula el intervalo asociado a cada una de ellas. Este paso es necesario hacerlo la primera vez, pero en las sucesivas vueltas se puede precalcular mientras se genera la secuencia de salida.
- Se recorre toda la secuencia buscando las posiciones en las que aparece por primera vez una dependencia (se abre su intervalo) después de que otra dependencia hubiese sido leída por última vez (su intervalo se cerrara).
- En cada subsecuencia en la que los puntos encontrados en el paso previo divide la entrada, o en la secuencia entera si no existe ningún punto, se busca la mayor subsecuencia de dependencias tales que todas las dependencias iguales a ella están contenidas en la subsecuencia y ésta contiene como máximo una dependencia que no cumple la condición. A la subsecuencia máxima le llamaremos *grupo*, las dependencias que cumplen la condición diremos que son de *tipo A* y las que no de *tipo B*. Hacer notar que estos grupos pueden no existir.
- A cada grupo encontrado se le asocia una nueva dependencia como representante que será igual a:
  - La dependencia de tipo B que está contenida en el grupo.
  - O en caso contrario, la última dependencia que abrió su intervalo antes del grupo. Si no existe dicha dependencia se usará la superdependencia S.
- La secuencia de salida es el resultado de sustituir cada grupo por su representante en la secuencia de entrada dejando el resto de dependencias en sus posiciones originales.

Ejemplo 4.1. En la ilustración 4.6 se muestra el árbol de dependencias resultante tras procesar la espina principal y el grupo que contiene a D_8* del ejemplo visto en la sección anterior. Para obtener el árbol final se debe sustituir los representantes D_6* por el resultado de aplicar el algoritmo para árboles simples a las secuencias correspondientes. Los subárboles que se obtienen figuran en la ilustración 4.7.

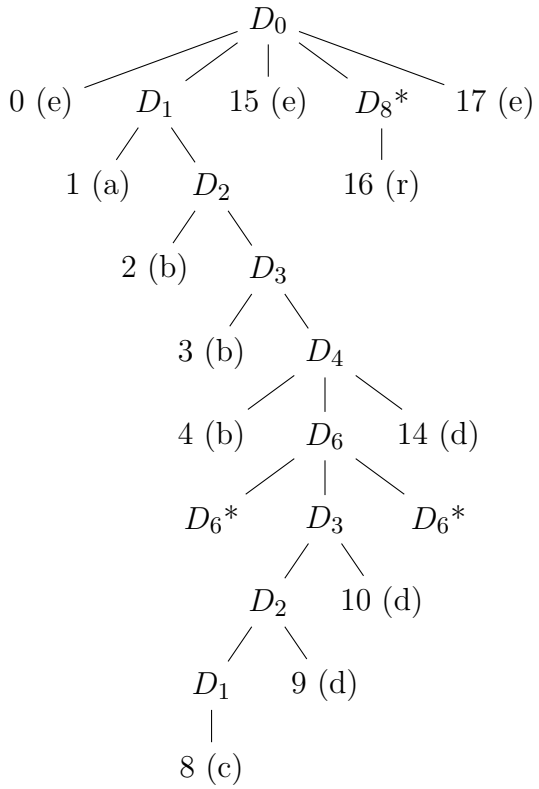

Ilustración 4.6 Árbol de dependencias de la espina principal de *eabbbtptcddvpvdere*.

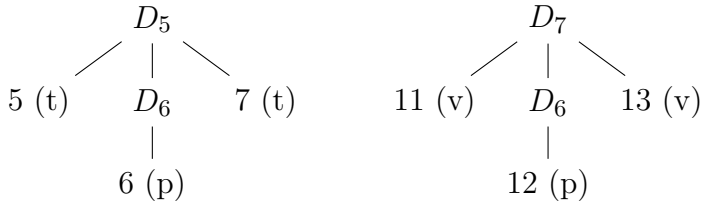

Ilustración 4.7 Subárbol de dependencias de *tpt* y *vpv*.

En el anexo D.2 se detalla una implementación en pseudocódigo junto a las estructuras de datos utilizadas. Una implementación 'real' ha sido incluida e integrada en el entorno de análisis y visualización de las 2-estructuras, la cual permite generar y visualizar los árboles de dependencias complejos a partir de una cadena con dependencias. También se puede visualizar la composición jerárquica de las distintas espinas de manera similar a lo mostrado en la ilustración 4.5.

.

5 Conclusiones

En este capítulo se presenta un resumen del conjunto del trabajo realizado y se describen las posibilidades de continuación. También se ofrece una valoración personal sobre el desarrollo del proyecto.

5.1. Resultados del proyecto

Este proyecto nació con la doble finalidad de proveer una herramienta para el estudio de las 2-estructuras y sus aplicaciones y de establecer resultados prometedores que relacionaran las 2-estructuras y las MCSG, objetivos que han sido cumplidos satisfactoriamente.

El trabajo realizado por el autor puede ser resumido en los siguientes tres puntos:

- El diseño y la implementación de un paquete de software que contiene tanto una biblioteca para futuros programas que requieran del análisis, la transformación y la visualización de las principales estructuras involucradas en la teoría de las 2-estructuras como un entorno de trabajo para usuarios finales que investiguen sobre el tema.

- La formulación de una extensión de las grámaticas de núcleo que dota a las cadenas generadas de una estructura que sirve para realizar comparaciones con otras gramáticas debilmente equivalentes, que además preserva el uso de las operaciones originales y que se fundamenta en la teoría de las 2-estructuras.

- El diseño de un algoritmo capaz de generar una gramática TAG a partir de una cadena (o frase del lenguaje natural) con dependencias anidadas y cruzadas.

Como resultado adicional se prevee la elaboración de un artículo de investigación cuyo germen será el presente proyecto en colaboración con el grupo de investigación del profesor Balcázar en la Universidad de Cantabria.

5.2. Trabajo futuro

Parte del objetivo con el que se ha construido el paquete de software es el de servir de herramienta base para futuras investigaciones en relación con las 2-estructuras. Sin embargo, es imposible adelantarse a todos los posibles usos que se le pueda dar. Es por ello que será necesario seguir complentando y adaptando el trabajo aquí realizado para ajustarse a las necesidades de trabajos venideros.

En relación con la extensión de las gramáticas de núcleo presentada, se propone diseñar un algoritmo que, en vez de generar una gramática TAG, genere una gramática HG (en su versión extendida) a partir de una cadena con dependencias. Después

se podría comparar ambos algoritmos y estudiar sus semejanzas y diferencias en cuanto a los métodos usados y las estructuras generadas.

Otro punto interesante a analizar en profundidad es ver cuanto se parece las gramáticas generadas a partir de una palabra a las que uno construiría con la intención de poder derivarla. La experiencia tenida durante el trabajo muestra que aunque muchas veces éstas coinciden, muchas otras no lo hacen.

Por último, se propone como siguiente paso natural, una vez que se es capaz de generar una gramática a partir de una sola palabra, el buscar la mejor forma de unificar todas las gramáticas generadas a partir de palabras pertenecientes a un mismo lenguaje.

5.3. Valoración personal

La realización de este proyecto fin de carrera como colofón a mis estudios ha sido una experiencia muy gratificante y enriquecedora.

Por una parte por la doble naturaleza del proyecto, que se compone de una parte más teórica y una parte más práctica.

En la parte práctica tenemos el desarrollo del paquete de software, donde he podido aplicar muchos de los conocimientos de ingeniería y programación adquiridos durante la carrera.

En la parte teórica está el estudio de las 2-estructuras, del procesamiento del lenguaje natural y las MSCG, temas desconocidos o de los que tenía un conocimiento muy superficial y cuya exploración me ha resultado muy interesante. En particular la teoría de las 2-estructuras, que ha influido en mi forma de abordar nuevos problemas con su visión 'puramente' matemática.

Y por otra parte por la libertad que he tenido y el apoyo que he recibido para desarrollar el proyecto.

Bibliografía

[1] A. Ehrenfeucht and G. Rozenberg. "Theory of 2-structures. Part I: Clans, basic subclasses, and morphisms". Theor. Comput. Sci., 70:277–303, 1990.

[2] A. Ehrenfeucht and G. Rozenberg. "Theory of 2-structures. Part II: Representation through labeled tree families". Theor. Comput. Sci., 70:305–342, 1990.

[3] A. Ehrenfeucht and G. Rozenberg. "Angular 2-structures". Theor. Comput. Sci., 92:227–248, 1992.

[4] A. Ehrenfeucht and G. Rozenberg. "T-Structures, T-functions, and texts". Comput. Sci., 116:227–290, 1993.

[5] A. Ehrenfeucht, T. Harju and G. Rozenberg. "The theory of 2-estructures. A Framework for Decomposition and Transformation of Graphs". World Scientific, 1999.

[6] D. Jurafsky, J.H. Martin: "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition". Prentice Hall, 2009.

[7] M. A. Alonso. "Interpretación tabular de autómatas para lenguajes de adjunción de árboles", Tesis doctoral, Universidad de La Coruña, España, 2000.

[8] K. Vijay-Shanker, D.J. Weir. "The Equivalence Of Four Extensions Of Context-Free Grammars". Mathematical Systems Theory, 27:511–546, 1994.

[9] J.M. Vergés. "La no-independencia del contexto de los lenguajes naturales". Lenguajes naturales y lenguajes formales: actas del XII congreso de los lenguajes naturales y lenguajes formales, pp. 87—102, 1996.

[10] C. Pollard. "Generalized Phrase Structure Grammars, Head Grammars and Natural Language". PhD thesis, Stanford Ubiversity, 1984.

[11] D.J. Weir. "Characterizing Mildly Context-Sensitive Grammar Formalisms". PhD thesis, University of Pennsylvania, 1988. Available as Technical Report MS-CIS-88-74 of the Department of Computer and Information Sciences, University of Pennsylvania.

[12] D.J. Weir, K. Vijay-Shanker and A.K. Joshi. "The relantionship Between Tree Adjoining Grammars And Head Grammars". Proceedings of the 24th annual meeting on Association for Computational Linguistics, 67–74, 1986.

[13] J.Q. Walker II. "A node-positioning algorithm for general trees". Softw. Pract. Exper., 20:685–705, 1990.

[14] C. Buchheim, M. Jünger and S. Leipert. “Improving Walker’s Algorithm to Run in Linear Time”, Graph Drawing. Lecture Notes in Computer Science, 2528:347–364, 2002.

[15] A.K. Joshi. “Processing Crossed and Nested Dependencies: An automaton Perspective on the Psycholinguistic Results”. Language and Cognitive Processes, 1989.

[16] Bernard A. Galler and Michael J. Fischer. “An improved equivalence algorithm”. Communications of the ACM, 7:301–303, 1964.

[17] Plataforma Java SE. http://www.oracle.com/technetwork/java/javase/ (Último acceso: 24/08/2010).

[18] JGraph. http://www.jgraph.com (Último acceso: 24/08/2010).

[19] JUNG. http://jung.sourceforge.net/ (Último acceso: 24/08/2010).

[20] Eclipse. http://www.eclipse.org/ (Último acceso: 24/08/2010).

[21] Subversion. http://subversion.apache.org/ (Último acceso: 24/08/2010).