Conciliación del Conocimiento mediante retracción conservativa

Memoria presentada por
Alejandro Trujillo Caballero

Tutor: Gonzalo A. Aranda Corral

Agradecimientos

A mi familia por su apoyo y comprensión durante estos años de estudio.

A mis profesores por su esfuerzo y dedicación, en especial a Gonzalo A. Aranda por su ayuda y atención durante la realización de este proyecto.

Índice general

Introducción

Actualmente al aplicar Análisis Formal de Conceptos u otros procedimientos que puedan trabajar con bases de conocimiento basadas en implicaciones tratar con grandes cantidades de datos puede ser complicado desde el punto de vista computacional.

Una forma de abordar este problema puede ser reducir el tamaño y complejidad de una base de conocimiento antes de empezar a trabajar con ella, por ejemplo eliminando información no relevante para lo que se intenta analizar.

Desde un punto de vista más humano, una persona es capaz de manejar simultáneamente varios ámbitos de información, pero es capaz de separarlos a la hora de tomar decisiones según la situación en la que se encuentra. Por ejemplo, una persona modera su comportamiento en su hogar de forma diferente que su entorno de trabajo, de forma que mientras trabaja tiene en cuenta factores a los que no atiende en su hogar aunque este tomando una decisión sobre el mismo asunto. Tomar un café normal o descafeinado, puede tener diferentes respuestas en función del entorno o de la hora.

El ser humano es capaz de razonar ignorando reglas que se aplican al problema que esta manejando en función del contexto. Podemos conseguir algo similar en sistemas basados en reglas mediante Retracción Conservativa.

Análisis formal de conceptos

El Análisis Formal de Conceptos (FCA) es una teoría matemática que se aplica en campos como la minería de datos con el objetivo de formalizar y estudiar las nociones de concepto y jerarquía de conceptos, su extracción y análisis.

Al aplicar FCA se parte generalmente de una tabla en la que cada fila es un objeto, cada columna un posible atributo de los objetos y cada celda indica si ese objeto posee o no el atributo correspondiente. A esto se le conoce como el **Contexto formal**.

A partir del contexto pueden obtenerse los conceptos formales, explicado de forma coloquial un concepto formal es un subconjunto de los atributos, todo objeto que posea esos atributos estará contenido en el concepto. Por ejemplo en una tabla que contiene animales, todos los animales que posean los atributos *Vive en el agua* y *Vive en la tierra* estarán contenidos en un mismo concepto que sabemos que es *Animal anfibio*.

Este conocimiento puede representarse de diferentes maneras, algunas de las más utilizadas son:

- **Retículo de Conceptos**: un grafo que representa los conceptos como nodos y muestra una relación de orden parcial entre ellos (un concepto puede estar contenido dentro de otro).

- **Implicación de atributos**: este método de representación consiste en escribir un conjunto de implicaciones entre los atributos del contexto de forma que lo que se expresa es "Todo objeto que satisface estos atributos, también satisface estos otros".

Retracción de teorías lógicas

A continuación se presentan diferentes conceptos de lógica matemática desde un punto de vista "informal" necesarios para la correcta comprensión del resto de esta memoria.

Aunque en nuestro caso el trabajo posterior se centra sobre el tratamiento de implicaciones, estas definiciones son genéricas y son ciertas para la lógica matemática completa.

Extensión y Retracciones Conservativas

En lógica decimos que una teoría T es una **extensión conservativa** de una teoría T' (o que T' es una **retracción conservativa** de T) si toda consecuencia de T en el lenguaje de T' es también consecuencia de T'.

La conclusión que se puede extraer de esta definición interesante para nuestro trabajo es, explicado de una forma más coloquial, que partiendo de una teoría escrita en un lenguaje podemos encontrar una teoría escrita en otro lenguaje más reducido. Todo lo que es cierto en esta nueva teoría también es cierto en la primera y todo lo que es cierto en la primera (y puede expresarse en el lenguaje de la segunda) es cierto. Esta nueva teoría será una retracción conservativa de la primera.

Esto nos permite por ejemplo, poder demostrar algo en una teoría de tamaño más reducido y fácil de trabajar, sabiendo que el resultado será válido para cualquier extensión conservativa de esa teoría.

Aplicación de la retracción

Al trabajar en FCA podemos representar el conocimiento como conjuntos de implicaciones entre atributos y podemos utilizar los conceptos de extensión y retracción conservativa para facilitar el trabajo sobre esto conjuntos.

Aplicar retracción de implicaciones al Análisis Formal de Conceptos nos permite obtener ese "filtrado de conocimiento" humano sobre una base de reglas, eliminando de la base todos los atributos que no tengan relevancia para nuestro estudio y por tanto reduciendo la complejidad del sistema pero a la vez manteniendo la validez de toda deducción o consecuencia lógica que obtengamos.

Objetivo

El objetivo de este proyecto es la implementación de un **Retractor de Implicaciones** capaz de actuar sobre bases de conocimiento completas.

Para ello se realizarán diferentes pasos:

Framework de lógica matemática En primer lugar se implementará un framework que permita trabajar con entidades lógicas (fórmulas, implicaciones, clausulas, etc) que posteriormente facilitará la implementación y testeo del retractor.

Retractor de Implicaciones Se realizará una implementación de un algoritmo retractor de implicaciones básico.

Optimizaciones Por último se añadirán diferentes optimizaciones a la implementación básica del retractor.

Explicacion ejemplo??

Enlace o referencia a la web del lenguaje o no es necesario?

Elección del lenguaje de programación

La implementación se ha realizado en Scala.

Scala es un lenguaje orientado a objetos y funcional que se ejecuta sobre la JVM (Java Virtual Machine).

Las razones por las que se ha escogido son:

- Funcional: Los lenguajes funcionales están planteados desde un punto vista muy cercano a las matemáticas por lo que implementar conceptos de lógica matemática en este tipo de lenguajes es más directo.
- JVM: Al ser un lenguaje que se ejecuta en el entorno Java, permite de forma fácil general un ejecutable que puede funcionar en cualquier plataforma.

Parte I

FCA

Análisis formal de conceptos

En este apartado se describe de forma más formal y detallada en que consiste el **Análisis formal de conceptos**, sus componentes y las dos formas de representación mencionadas en la introducción (Retículo de conceptos e Implicación de atributos).

Contextos formales y concepto formal

La unidad básica de representación del conocimiento en FCA es el **Contexto formal**

Contexto Formal Un contexto formal ($\mathbb{C}$) es una tripleta formada por un conjunto de objetos (O), un conjunto de atributos (A) y una relación binaria (I) entre objetos y atributos ($I \subseteq O \times A$)

Esto normalmente se representa como una tabla donde las filas representan los objetos, las columnas los atributos y una cruz en la fila a de la columna o significa que el objeto o posee el atributo a. Esto puede expresarse como $o \in O, a \in A, (o, a) \in I$.

Por ejemplo, en el contexto mostrado en el cuadro 1 se describe si determinados países pertenecen a organizaciones internacionales (UNP, CT, G8, EU, UN) y si poseen armas nucleares (NW).

El objetivo de FCA es extraer los conceptos existentes a partir del contexto formal, para ello debemos definir primero la operación básica dentro de la teoría de FCA, el **operador derivación**.

Paises	NW	UNP	CT	G8	EU	UN
USA	×	×		×		×
Alemania				×	×	×
Francia	×	×		×	×	×
Reino Un.	×	×		×	×	×
Turquía						×
Qatar			×			×
Italia			×	×	×	×

Cuadro 1: Contexto formal de paises

Operador derivación La derivada de un conjunto de atributos $At \subseteq A$ se define como:

$$At' = o \in O | \forall a \in At : (o, a) \in I$$

Análogamente, la derivada de un conjunto de objetos $Ob \subseteq O$ se define como:

$$Ob' = a \in A | \forall o \in Ob : (o, a) \in I$$

De forma coloquial, la derivada de un conjunto de atributos es el conjunto de objetos que poseen todos esos atributos y la derivada de un conjunto de objetos es el conjunto de atributos comunes para todos esos objetos.

Estos operadores forman una *conexión de Galois* y verifican ciertas propiedades que no entraremos a analizar aquí.

Referencia para ampliar esto

Finalmente a partir de la definición de derivación, podemos definir un **concepto formal**.

Concepto formal Un concepto formal de un contexto $\mathbb{C} = (O, A, I)$ es un par (Ob, At) que cumple

$$At' = Ob \quad y \quad Ob' = At$$

Dado el concepto $C = (Ob, At)$, se denomina:

Extensión (Ext) del concepto Conjunto de objetos Ob que lo componen.

Intensión (Int) del concepto Conjunto de atributos At del concepto.

Con esto podemos ver que un concepto esta formado por un conjunto de atributos y un conjunto de objetos, tales que los objetos comparten los atributos del conjunto y este conjunto solo contiene los atributos que comparten los objetos.

Algunos de los conceptos que pueden extraerse del contexto formal del cuadro 1 son:

1. ({Alemania, Reino Un, Francia, Italia}, {EU, G8, UN})
2. ({USA, Francia, Reino Un}, {NW, UNP, G8, UN})
3. ({USA, Alemania, Francia, Reino Un, Italia}, {G8, UN})
4. ({Turquía, USA, Alemania, Qatar, Francia, Reino Un, Italia}, {UN})

Retículo de conceptos

Antes de definir en que consiste un retículo de conceptos, necesitamos definir las relaciones entre conceptos. La definición de concepto vista anteriormente nos permite definir un orden parcial entre los mismos:

Relación de orden Sea $C_1 = (O_1, A_1)$ y $C_2 = (O_2, A_2)$ dos conceptos pertenecientes a un contexto $\mathbb{C} = (O, A, I)$. Definimos la relación de orden $\preceq$ como:

$$C_1 \preceq C_2 \Longleftrightarrow O_1 \subseteq O_2 \;\; (\Leftrightarrow A_2 \subseteq A_1)$$

Se dice que C_1 es **subconcepto** de C_2 (o C_2 es **superconcepto** de C_1). Explicado de forma menos formal, un concepto es subconcepto de otro cuando su conjunto de objetos es un subconjunto de los de el segundo concepto (o lo que es lo mismo, el conjunto de atributos del segundo es un subconjunto de los del primero.).

Esto establece una relación jerárquica de orden parcial entre los conceptos formales. El conjunto de todos los conceptos de un contexto junto con esta relación de orden forman un retículo completo que se denomina **Retículo de conceptos** y se denota como $\mathcal{L}(\mathbb{C})$.

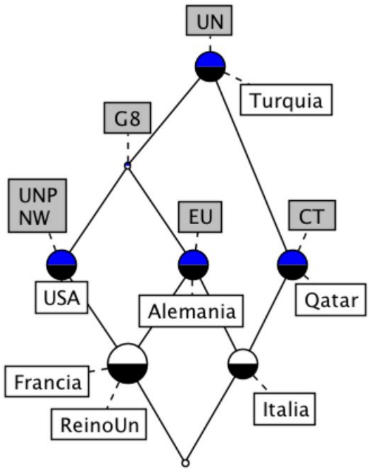

Figura 1: Imagen del retículo

Representando gráficamente el retículo podemos leer fácilmente los objetos, atributos y relaciones y ayuda a comprender la estructura de los datos y el contexto. El retículo se representa como un grafo en el que los nodos representan conceptos formales y mediante el etiquetado de los mismos con los atributos y objetos puede calcularse de forma rápida la extensión o intensión de cualquiera de ellos simplemente siguiendo las lineas hacia arriba (intensión) o hacia abajo (extensión).

En la figura 1 puede verse el retículo correspondiente al contexto formal del cuadro 1.

cono, atributos propios...? es necesario?

Implicación de atributos

Otra forma de especificar un contexto formal (y la que nos interesa para nuestro trabajo) es escribiendo un conjunto de implicaciones entre los atributos del contexto de forma que el contexto pueda ser totalmente reconstruido partiendo del conjunto de implicaciones.

Una implicación es una restricción sobre los atributos con la forma:

$$\{a_i, ..., a_j\} \to \{a_k, ..., a_m\}$$

Lo cual puede leerse como "Todo objeto que posee los atributos $\{a_i, ..., a_j\}$ también posee los atributos $\{a_m, ..., a_k\}$".

La definición formal de este concepto es:

Implicaciín entre atributos Sea $\mathbb{C} = (O, A, I)$ un contexto formal. Una implicación de atributos es un par de conjuntos $L, R \subseteq A$, normalmente escritos $L \to R$. Una implicación $L \to R$ es válida en $\mathbb{C}$ si para todo objeto de $\mathbb{C}$ que tiene todos los atributos de L también tienen todos los atributos de R. Todas las implicaciones extraídas de un contexto $\mathbb{C}$ se denotan como $Imp(\mathbb{C})$.

Algunas implicaciones que pueden extraerse del contexto del cuadro 1 son:

- { } → UN
- CT, G8, UN → EU
- EU, UN → G8

Estas implicaciones entre atributos tienen las mismas propiedades que las pertenecientes a la lógica proposicional por lo que podemos utilizar esta última para realizar razonamiento sobre esta representación del contexto formal y transformarlo de forma acorde a las reglas de la lógica matemática. Esto nos permite aplicar la **retracción conservativa** mencionada en la introducción sobre conjuntos de implicaciones de atributos generados tras la aplicación de FCA a un contexto.

Parte II

RETRACCIÓN

Retracción conservativa

nombre de capitulo adecuado?

Tras sentar las bases conceptuales de la retracción y extensión conservativa en la introducción de esta memoria, en este apartado especificaremos como se aplican estos conceptos de forma concreta a la lógica proposicional ya que esta es la que nos permite tratar las implicaciones que obtenemos de FCA y razonar sobre ellas.

Retracción conservativa en lógica proposicional

La definición de retracción y extensión aplicada a la lógica proposicional es la misma que en el caso genérico pero podemos especificar los conceptos de teoría y lenguaje. En este caso siendo el lenguaje un conjunto de atributos y una teoría un conjunto de proposiciones que utilizan dichos atributos (a lo cual nos referimos como base de conocimiento).

En principio el concepto de proposición puede englobar diferentes tipos: negaciones, implicaciones, conjunciones... Nosotros trabajaremos únicamente con implicaciones, pero esto no afecta a los conceptos aquí explicados ya que nuestro trabajo sigue estando contenido dentro de la lógica proposicional, simplemente nos limitamos a trabajar con un subconjunto de la misma.

Extensión y retracción conservativa Partiendo de un base de conocimiento $\mathcal{L}$ decimos que $\mathcal{L}'$ es una **extensión conservativa** de $\mathcal{L}$ si $\mathcal{L}'$ contiene al menos todas los atributos presentes en $\mathcal{L}$ y además todo lo que cierto en $\mathcal{L}$ lo es también en $\mathcal{L}'$. Podemos referirnos a lo mismo diciendo que $\mathcal{L}$ es una **retracción**

conservativa de $\mathcal{L}'$.

Viendo esto, podemos ver que partiendo de una base de conocimiento podemos eliminar atributos de la misma y obtener una retracción conservativa con menor número de atributos. Para esto definimos el **operador de olvido**

Operador de olvido Un **operador de olvido** del un atributo p es aquel que aplicado sobre dos proposiciones devuelve un nuevo conjunto de proposiciones equivalentes pero que no contienen el atributo p.

Es fácil demostrar que este operador aplicado sobre las combinaciones de las proposiciones de una base de conocimiento nos permite obtener una retracción conservativa de dicha base que no contiene el atributo p.

Relación con FCA

Ya que nuestro objetivo es trabajar con implicaciones de atributos obtenidas en FCA podemos especificar aun más los conceptos descritos en este apartado.

Extensión y retracción conservativa Sea $\mathbb{C} = (O, A, I)$ un contexto formal y $\mathcal{L} = Imp(\mathbb{C})$ el conjunto de implicaciones derivado de ese contexto. Se dice que $\mathcal{L}$ es una **extensión conservativa** de $\mathcal{L}'$ (o que $\mathcal{L}'$ es una **retracción conservativa** de $\mathcal{L}$) si:

$$\mathcal{L} \models \mathcal{L}' \quad y \quad \forall L \in Imp(att(\mathcal{L}'))[\mathcal{L} \models L \Longrightarrow \mathcal{L}' \models L]$$

Lo cual significa que dado un conjunto de implicaciones y una extensión conservativa del mismo, cualquier implicación construida utilizando los atributos del conjunto inicial sera cierta en la extensión si lo es en el conjunto original.

Una vez que conocemos esta definición queremos ser capaces de construir retracciones conservativas de los conjuntos de implicaciones obtenidos por medio de FCA. Para ello necesitamos definir un operador de olvido. Se pueden definir diferentes operadores de olvido todos ellos válidos desde el punto de vista lógico, pero no todos son válidos para nuestro problema.

Puesto que estamos trabajando sobre la lógica proposicional cualquier operador de olvido válido funcionará con nuestras implicaciones, pero el conjunto resultante de su aplicación no tiene por qué estar limitado a contener únicamente implicaciones.

Por ello utilizaremos un operador de olvido específicamente creado para esta problemática y cuya validez se demuestra en . referencia

Operador de olvido para implicaciones Sea $C_i = Y_1^i \to Y_2^i$ una implicación tal que $Y_1^i \cap Y_2^i = \emptyset$. El operador $\partial_p(C_1, C_2)$ es un operador de olvido del atributo p:

$$\partial_p(C_1, C_2) = \begin{cases} \{C_1, C_2\} & p \notin att(C_1) \cup att(C_2) \\ \{C_2\} & p \in Y_1^1, p \notin att(C_2) \\ \{Y_1^1 \to (Y_2^1 \setminus p), C_2\} & p \in Y_2^1, p \notin att(C_2) \\ \{\top\} & p \in (Y_1^1 \cap Y_1^2) \cup (Y_2^1 \cap Y_2^2) \\ \{Resolvente_p(C_1, C_2)\} & p \in Y_1^2 \cap Y_2^1 \end{cases}$$

donde

$$Resolvente_p(C_1, C_2) := \{Y_1^1 \to Y_2^1 \setminus \{p\}, Y_1^1 \cup (Y_1^2 \setminus \{p\}) \to Y_2^2\}$$

Este operador es simétrico, es decir $\partial_p(C_1, C_2) = \partial_p(C_2, C_1)$ y tal como indicamos anteriormente, aplicándolo sobre las combinaciones de las implicaciones de un conjunto podemos eliminar por completo un atributo del mismo.

ejemplo de aplicacion del operador?

Parte III

IMPLEMENTACIÓN Y MONTAJE

Implementación y montaje

En este apartado se explican los detalles de la implementación realizada del programa así como su funcionamiento.

El proceso de implementación se ha realizado en varias fases, en primer lugar se ha construido un pequeño framework o librería que permita trabajar con lógica matemática de cara a poder utilizar estas herramientas para facilitar la codificación del programa final y su testeo. Las funcionalidades contenidas en esta librería están pensadas para utilizarse como herramienta de desarrollo y por tanto un usuario no puede acceder a ellas de forma directa desde el programa final.

Una vez construida esta base, se ha realizado la implementación del algoritmo de retracción así como varias optimizaciones sobre el mismo y finalmente se ha empaquetado como aplicación java de linea de comando en un archivo .jar.

Framework de programación lógica

Para implementar los diferentes conceptos de lógica de esta librería se ha tomado como base otra anterior, realizada en Haskell por , sobre la que se han realizado algunas modificaciones, mejoras y adaptaciones. Ya que Haskell es un lenguaje puramente funcional, el código se puede transformar a Scala de forma relativamente simple ya que aunque la sintaxis no es similar los conceptos con los que trabaja son los mismos. referencia

A continuación se describen los apartados más importantes de la librería y su funcionamiento.

Proposiciones

Los componentes más básicos de la librería son las estructuras de datos que permiten crear, almacenar y operar sobre proposiciones lógicas.

Partiendo de un tipo *Prop* (Proposición) y utilizando herencia y polimorfismo definimos siete tipos de datos con los que se puede construir cualquier proposición lógica:

Constante Representa un valor booleano de verdadero o falso.

Atom Representa una proposición que solo consta de una variable. Por ejemplo p o q serían representados mediante una instancia de tipo Atom.

Negación (Neg) Representa la negación de otra proposición ($\neg$).

Conjunción (Conj) Representa el operador lógico $\wedge$ (AND) entre dos proposiciones.

Disyunción (Disj) Representa el operador lógico $\vee$ (OR) entre dos proposiciones.

Implicación (Impl) Representa la implicación entre dos proposiciones. En lógica representado por $\rightarrow$.

Equivalencia (Equi) Representa la equivalencia entre proposiciones. En lógica representado como $\leftrightarrow$.

Con estas siete estructuras puede representarse cualquier proposición lógica, por ejemplo:

$$\neg p \vee (q \rightarrow r) = Disj(Neg(Atom(p)), Impl(Atom(q), Atom(r)))$$

Tras la implementación de estas estructuras se realizó la implementación de un pequeño DSL (Domain specific language) que facilita la escritura de este tipo de fórmulas mediante el uso de algunas funciones y operadores. El ejemplo anterior escrito en este lenguaje quedaría:

$$no(q)\ OR\ (q \mathrel{-}> r)$$

Otro concepto que necesita ser modelado es el de **Interpretación**, una interpretación es una asignación de valores verdadero o falso a las variables de una proposición, si tras la sustitución de los valores indicados por la interpretación la evaluación de la proposición es cierta, decimos que la interpretación es un **modelo** de dicha proposición.

Esta entidad se maneja en la librería como un diccionario (un almacenamiento clave - valor) en el que se guardan las asignaciones a cada variable y la ausencia de alguna como clave se considera como que esa variable se asigna como falsa.

Finalmente, se implementaron diferentes operaciones como obtener si una interpretación es modelo de una proposición o obtener todos los modelos existentes para una proposición, así como funciones que permiten operar directamente sobre conjuntos de proposiciones.

Hasta que punto entrar en detalle en esta parte?

Formas normales y cláusulas

Una misma proposición puede representarse de diferentes formas, es usual en lógica trabajar sobre formas normalizadas de las proposiciones por lo que la librería permite la transformación de proposiciones a diferentes formas normales (forma normal negativa, conjuntiva y disyuntiva)

Una de las formas de representación más utilizadas es la **cláusula** y por ello se encuentra representada en la librería por un tipo de datos propio. Una cláusula es una proposición que solo contiene literales y disyunciones (o conjunciones aunque en nuestro caso este segundo modelo no se ha implementado), un **literal** puede ser un símbolo atómico (*Atom* en la representación de proposiciones) o la negación de un símbolo atómico.

De esta forma, modelando el tipo literal, podemos representar una clausula como un conjunto de literales, los cual es computacionalmente más fácil de manejar que la estructura recursiva que modela las proposiciones.

Una de las operaciones más importantes que suele realizarse al trabajar con cláusulas es la de **resolvente**. Esta operación ya ha sido nombrada en la definición que dimos anteriormente para el operador de olvido para implicaciones. La resolvente de dos clausulas respecto a un literal es una nueva cláusula con la misma semántica que las anteriores pero que no contiene dicho literal.

La librería permite operaciones como el cálculo de resolventes de diferentes tipos así como cálculos de todas las resolventes posibles de conjuntos de clausulas, etc.

Razonamiento

Hasta ahora tenemos representación y operaciones sobre entidades lógicas pero la parte más interesante es poder realizar razonamiento lógico sobre estas representaciones.

La librería permite el cálculo diferentes conceptos como validez, consistencia, consecuencia lógica, etc... Mediante el uso de diferentes métodos que operan sobre las diferentes representaciones (proposiciones y cláusulas). Algunos de estos sistemas de razonamiento son:

Fuerza bruta La librería contiene implementaciones exhaustivas de algunos de los conceptos mediante la comprobación de todas las posibilidades.

Tableros semánticos Es un método que actúa sobre conjuntos de proposiciones para obtener todos los modelos del conjunto. Realizando modificaciones sobre el conjunto puede utilizarse para calcular diferentes conceptos, por ejemplo podemos probar que una proposición es un teorema si el conjunto de modelos de su negación es vacío, y este conjunto puede calcularse mediante tableros semánticos.

Davis-Putnam Es un algoritmo que actúa sobre conjuntos de cláusulas para comprobar su satisfacibilidad, al igual que en el caso anterior pueden obtenerse diferentes resultados aplicándolo sobre modificaciones del conjunto inicial.

Cálculo de secuentes Es un método de razonamiento que funciona sobre proposiciones pero que no solo nos permite probar fórmulas lógicas sino que nos indica el proceso de la prueba paso a paso, de forma que cada linea de la demostración utiliza las lineas anteriores de la misma.

Con estos métodos podemos aplicar razonamiento sobre cualquier conjunto de proposiciones o cláusulas que obtengamos, permitiéndonos por ejemplo comprobar si el conjunto de implicaciones generado al aplicar el algoritmo de retracción es equivalente al original.

Aqui pruebas pequeñas

Parte IV

EJEMPLO GRANDE (CONCILIACIÓN DELICIOUS)

Parte V

Conclusiones

Capítulo 1

Conclusiones

Parte VI

Bibliografía

Bibliografía