Aplicación de retracción conservativa al análisis formal de conceptos

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

Hoy en día se generan grandes cantidades de datos con los que normalmente se trabaja de forma informatizada. En ocasiones trabajar sobre estos conjuntos puede ser complicado y costoso desde el punto de vista computacional, sobre todo al intentar aplicar mecanismos de aprendizaje automático o razonamiento como el Análisis formal de conceptos.

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 contenida en el contexto de lo que se intenta analizar. Sin embargo, esta eliminación no puede realizarse de forma arbitraria, debe hacerse respetando la consistencia lógica de los razonamientos que se realizan posteriormente.

Desde un punto de vista más humano, una persona es capaz de manejar simultáneamente varios ámbitos de información, siendo 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 en 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 del día.

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.

Este "razonamiento bajo contexto" puede aplicarse de forma computacional a las nuevas tecnologías. Por ejemplo, en un teléfono móvil con potencia limitada pueden

utilizarse sensores (wifi, gps...) para aplicar filtros sobre un contexto y permitir ejecutar razonamiento para automatizar tareas trabajando sobre un sistema de tamaño reducido que el procesador del teléfono pueda manejar con soltura.

Objetivo

El objetivo de este proyecto es la implementación de un sistema eficaz capaz de reducir una base de conocimiento, preservando la consistencia lógica, para su posterior uso en razonamiento bajo contexto.

Para ello se realizarán diferentes pasos:

Framework lógico-matemático Implementación de un framework que permita trabajar con entidades lógicas (fórmulas, implicaciones, clausulas, etc) que posteriormente facilita la implementación y testeo del retractor.

Retractor de Implicaciones Implementación de un algoritmo retractor de implicaciones básico.

Optimizaciones Aplicación de propiedades matemáticas de la retracción para la optimización de la implementación anterior.

Parte I

FUNDAMENTO TEÓRICO

Retracción conservativa

Retracción de teorías lógicas

En este apartado se presentan diferentes conceptos de lógica matemática 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 terminos de lógica matemática se dice que una teoría T es una **extensión conservativa** de una teoría T' (o, lo que es lo mismo, T' es una **retracción conservativa** de T) si toda consecuencia lógica de T en el lenguaje de T' es también consecuencia lógica de T'.

Explicado de una forma más coloquial, esto significa 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.

Retracción conservativa en lógica proposicional

A partir de esta definición de la retracción y extensión conservativa, en este apartado especificaremos como vamos a aplicar estos conceptos de forma concreta a la lógica proposicional ya que esta es la que se utiliza para trabajar con las implicaciones que obtenemos de FCA y razonar sobre ellas.

Aplicando la definición de retracción de forma directa a la lógica proposicional podemos especificar algunos conceptos para optimizar la implementación. En este caso, el lenguaje es un conjunto de atributos y una teoría un conjunto de proposiciones que utilizan dichos atributos (a lo cual nos referimos como base de conocimiento).

Aunque el concepto de proposición engloba diferentes tipos: negaciones, implicaciones, conjunciones, etc... Nosotros vamos a trabajar únicamente con implicaciones. Esto no afecta a los conceptos aquí explicados ya que nuestro trabajo sigue estando contenido dentro de la lógica proposicional.

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 $\mathcal{L}'$. Del mismo modo podemos decir que $\mathcal{L}$ es una **retracción conservativa** de $\mathcal{L}'$.

Es decir, a partir de una base de conocimiento podríamos eliminar ciertos 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** de un atributo p es aquel que aplicado sobre dos proposiciones devuelve un nuevo conjunto de proposiciones equivalentes pero que no contienen el atributo p.

Si aplicamos este operador sobre una base de conocimiento nos permite obtener una nueva base de conocimiento que es una retracción conservativa de la anterior y no contiene el atributo p.

Para explicar este operador, en primer lugar hablaremos de la interpretación polinomial de la lógica proposicional.

Lógica proposicional y el anillo $\mathbb{F}_2[x]$

Existen teorías probadas que permiten transformar proposiciones lógicas a polinomios, permitiendo el uso de herramientas algebraicas para resolver problemas lógicos.

Esta transformación da como resultado polinomios del anillo $\mathbb{F}_2[x]$, es decir, polinomios de grado 1 cuyos coeficientes solo pueden ser 0 o 1.

Las reglas utilizadas son:

$\pi : Prop \to \mathbb{F}_2[x]$ para transformar proposiciones a polinomio, definida por las ecuaciones:

- $\pi(\bot) = 0$, $\pi(p_i) = x_i$, $\pi(\neg F) = 1 + \pi(F)$
- $\pi(F_1 \wedge F_2) = \pi(F_1) \cdot \pi(F_2)$
- $\pi(F_1) \vee F_2) = \pi(F_1) + \pi(F_2) + \pi(F_1) \cdot \pi(F_2)$
- $\pi(F_1 \to F_2) = 1 + \pi(F_1) + \pi(F_1) \cdot \pi(F_2)$
- $\pi(F_1 \leftrightarrow F_2) = 1 + \pi(F_1) + \pi(F_2)$

$\Theta : \mathbb{F}_2[x] \to Prop$ para transformar de polinomio a proposición lógica, definida por:

- $\Theta(0) = \bot$, $\Theta(1) = \top$, $\Theta(x_1) = p_i$
- $\Theta(a \cdot b) = \Theta(a) \wedge \Theta(b)$
- $\Theta(a + b) = \neg(\Theta(a) \leftrightarrow \Theta(b))$

Más información sobre estas transformaciones puede encontrarse en [4].

A estos polinomios puede aplicarse cualquier operación algebraica, como por ejemplo la derivada.

Al derivar un polinomio en $\mathbb{F}_2[x]$ respecto a una variable dejando el resto constante, siempre obtendremos un nuevo polinomio que no contiene dicha variable (ya que si la variable aparece siempre tendrá exponente 1). Utilizando esta idea podemos buscar la forma de eliminar variables de nuestras fórmulas.

Podemos definir una derivada lógica de la forma:

$$\begin{array}{ccc} PForm & \xrightarrow{\partial} & PForm \\ \pi \downarrow & \# & \uparrow \Theta \\ \mathbb{F}_2[\mathbf{x}] & \xrightarrow{d} & \mathbb{F}_2[\mathbf{x}] \end{array}$$

Siendo,

$$\partial = \theta \circ d \circ \pi$$

Utilizando esto, tal como se demuestra en [2], puede definirse la **Regla de independencia** (o **regla** ∂) en fórmulas polinómicas que nos permite a partir de dos polinomios obtener uno nuevo que no contiene una variable concreta pero que es equivalente (desde el punto de vista lógico) a los iniciales. Esta regla puede utilizarse como un **operador de olvido** genérico.

Análisis formal de conceptos

El Análisis Formal de Conceptos (FCA) es una teoría matemática cuyo objetivo es formalizar las nociones de concepto y jerarquía de conceptos, su extracción y análisis.

Usualmente FCA parte de una tabla con información sobre diferentes objetos, la información contenida en esta tabla se procesa realizando un clustering jerárquico que agrupa los objetos en función de sus atributos. Estos clusters son los conceptos que se extraen de la información contenida en la tabla y pueden estar contenidos unos dentro de otros.

Por ejemplo si partimos de 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:

- **Contexto formal**: Es la información de la que parte FCA. Generalmente representado en forma de tabla.
- **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".

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

Cuadro 1: Contexto formal de paises

A continuación se describe formalmente en que consiste el **Análisis formal de conceptos** y sus componentes.

Contextos y conceptos formales

La unidad básica o fundamental de representación del conocimiento de la que parte FCA es el **Contexto formal**

Contexto Formal Un contexto formal (M) 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 esta relacionado con el atributo a. Esto puede expresarse como: siendo $o \in O$ y $a \in A$, se dice que o está relacionado con a si $(o, a) \in I$.

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).

Para extraer los conceptos existentes a partir del contexto formal se define primero la operación básica dentro de la teoría de FCA, el **operador derivación**.

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.

Finalmente a partir de la definición de derivación, se define un **concepto formal**.

Concepto formal Un concepto formal de un contexto $M = (O, A, I)$ 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 $C_2 = (O_2, A_2)$, tales que $O_1, O_2 \subseteq O$, $A_1, A_2 \subseteq A$, dos conceptos pertenecientes a un contexto $M = (O, A, I)$. 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 denotamos como $R(M)$.

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.

Implicación de atributos

Otra forma de especificar un contexto formal, y la que nos interesa para nuestro trabajo, es escribiéndolo como un conjunto de implicaciones entre atributos del contexto de forma que el contexto pueda ser reconstruido de forma equivalente partiendo del conjunto de implicaciones.

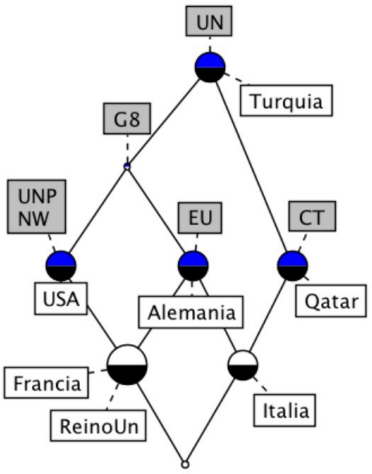

Figura 1: Imagen del retículo

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

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

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 una implicación de atributos en FCA es:

Implicación entre atributos Sea $M = (O, A, I)$ un contexto formal. Una implicación de atributos es un par de conjuntos $L, R \subseteq A$, normalmente escritos $L \rightarrow R$. Una implicación $L \rightarrow R$ es válida en M si para todo objeto de M que tiene todos los atributos de L también tienen todos los atributos de R (todo objeto que no tenga los atributos de L puede tener o no tener los atributos de R). Todas las implicaciones extraídas de un contexto M se denotan como $Imp(M)$.

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

- $\{\ \} \rightarrow$ UN
- CT, G8, UN $\rightarrow$ EU

- EU, UN → G8

Podemos interpretar estas fórmulas como si pertenecieran a la lógica proposicional y aplicar los algoritmos de razonamiento de la misma. 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.

ref

Para poder razonar sobre estos conjuntos de reglas se utilizan las reglas de Armstrong para implicaciones:

$\frac{}{X \to X}(Identidad)$ $\frac{X \to Y}{X \cup Z \to Y}(Extension)$ $\frac{X \to Y \quad X \cup Z \to W}{X \cup Z \to W}(Substitucion)$

Mediante la aplicación sucesiva de estas reglas puede realizarse razonamiento hacia delante partiendo de cualquier conjunto de implicaciones.

Retracción aplicada a 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 $M = (O, A, I)$ un contexto formal y $\mathcal{L} = Imp(M)$ 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:

Siendo $H = att(\mathcal{L}')$ el conjunto de atributos que aparecen en las implicaciones de $\mathcal{L}'$ y $K = impl(H)$ todas las implicaciones que pueden construirse con el conjunto de atributos H, se cumple que:

$$\mathcal{L} \models \mathcal{L}' \quad y \quad \forall L \in K \ \ [si\ \mathcal{L} \models L \Longrightarrow \mathcal{L}' \models L]$$

Lo cual significa que dado un conjunto de implicaciones y una retracción conservativa del mismo, cualquier implicación construida utilizando los atributos de la retracción sera cierta en la misma 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. 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 [1].

Operador de olvido para implicaciones Sea $C_i = Y_1^i \rightarrow Y_2^i$ 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:

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

donde

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

Aplicando este operador sobre todas las combinaciones de implicaciones de un conjunto podemos eliminar un atributo del mismo. Sin embargo este operador es simétrico, es decir $\partial_p(C_1, C_2) = \partial_p(C_2, C_1)$, lo cual nos permite realizar una importante optimización, podemos ignorar todas las combinaciones simétricas pasando de $O(n^2)$ a un $O(n \log(n))$.

Parte II

DISEÑO, IMPLEMENTACIÓN Y PRUEBAS

Diseño e implementación

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.

Elección del lenguaje de programación

La implementación se ha realizado en Scala, 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.

Framework de programación lógica

Para implementar los diferentes conceptos de lógica de esta librería se ha tomado como base otra anterior, realizada en Haskell (ver [3]), 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.

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 $\leftrightarrow$.

Un resumen del código que define estas estructuras (ignorando partes no relevantes):

```
type Symbol = String

sealed trait Prop
case class Const(boolean: Boolean) extends Prop
case class Atom(symbol: Symbol) extends Prop
case class Neg(prop: Prop) extends Prop
case class Conj(left: Prop, right: Prop) extends Prop
case class Disj(left: Prop, right: Prop) extends Prop
case class Impl(left: Prop, right: Prop) extends Prop
case class Equi(left: Prop, right: Prop) extends Prop
```

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:

```
val p = Atom("p")
val q = Atom("q")
val r = Atom("r")

val prop = no(q) OR (q -> 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.

```
type Interpretation = Map[Atom, Boolean]
```

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.

Por ejemplo, la función de la entidad Prop que calcula el significado de una proposición en base a una interpretación es:

```
sealed trait Prop {
...

   def meaning(interp: Interpretation): Boolean = this match {
      case Const(value) => value
      case Atom(symbol) => interp.getOrElse(Atom(symbol), false)
      case Neg(prop) => !prop.meaning(interp)
      case Conj(p, q) => p.meaning(interp) && q.meaning(interp)
      case Disj(p, q) => p.meaning(interp) || q.meaning(interp)
      case Impl(p, q) => (!(p meaning interp)) || (q meaning interp)
      case Equi(p, q) => {
         lazy val pm = p.meaning(interp)
         lazy val qm = q.meaning(interp)
         (!pm || qm) && (!qm || pm)
      }
   }

...
}
```

A partir de esto pueden implementarse de forma fácil otros conceptos como el de modelo o validez (tautología):

```
sealed trait Prop {
...
   def isModel(interpretation: Interpretation) = meaning(interpretation)

   def interpretations = symbols.subsets().map( interp => {
      interp.map(elem => (elem, true)).toMap
   })

   def models = interpretations.filter(isModel)

   def isValid = interpretations sameElements models
```

```
...
}
```

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:

Tabla de verdad (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.

Algoritmo retractor

Una vez construida la base de nuestra librería de lógica podemos empezar a implementar el algoritmo de retracción.

Retracción basada en polinomios

La primera aproximación realizada al algoritmo de retracción se basa en la transformación de proposiciones en polinomios sobre el anillo $\mathbb{F}_2[x]$.

Tal como se explica en el fundamento teórico, una vez las proposiciones se convierten a polinomios puede trabajarse matemáticamente con ellos, aplicar un algoritmo

de retracción basado en las derivadas de dichos polinomios y finalmente volver a convertir los polinomios a proposiciones.

Este método es válido, capaz de funcionar con cualquier tipo de proposiciones pero presenta un problema. Durante el proceso de conversión de proposición a polinomio y viceversa, la expresión de la proposición sufre alteraciones (se forman estructuras anidadas complejas) y aunque el resultado obtenido es correcto desde el punto de vista lógico, las formulas son muy complejas desde el punto de vista humano por lo que el análisis a simple vista de los resultados se vuelve prácticamente imposible.

Retractor de implicaciones

La versión definitiva del algoritmo, tal como se ha mencionado anteriormente, funciona exclusivamente sobre implicaciones con la estructura utilizada en FCA.

Para la implementación, en primer lugar se han utilizado unas versiones modificadas de las estructuras de datos *Implicación* y *Conjunción* de forma que estas no puedan contener otras proposiciones de forma recursiva:

Conjunción En este caso, la conjunción en lugar de representar el operador binario $\wedge$ representa la conjunción sobre una lista de variables de tamaño indefinido.

Implicación Una implicación se define como la implicación entre dos conjunciones.

Estas estructuras se han implementado de forma que su conversión a las estructuras proposicionales de la librería lógica es implícita y por tanto toda función de esta librería puede seguir aplicándose.

Utilizando estas estructuras, se ha realizado una implementación del operador de olvido (ecuación 1) explicado en el apartado de retracción.

Aplicando este operador sobre todas las combinaciones posibles de dos implicaciones de un conjunto podemos obtener uno nuevo que no contenga un atributo determinado. Sin embargo, debido a que el operador de olvido es simétrico las combinaciones con las mismas implicaciones y distinto orden son equivalentes, lo cual nos permite mejorar el orden de complejidad de este procedimiento.

Utilizando este operador, se implementa el algoritmo de retracción siguiendo el siguiente pseudocódigo:

```
1: function RETRACCION(implicaciones, variable)
2:     nuevasImplicaciones ← [ ]
3:
4:     for i ← 0 to implicaciones.size  do
5:         for j ← i to implicaciones.size do
6:             a ← implicaciones(i)
7:             b ← implicaciones(j)
8:             nuevasImplicaciones  ++  opOlvido(a, b, variable)
9:         end for
10:    end for
11:
12:    return nuevasImplicaciones
13: end function
```

Tras su aplicación obtenemos una nueva lista de implicaciones que no contiene la variable indicada por parámetro (una **retracción conservativa** del conjunto).

Este algoritmo tiene un orden temporal de ejecución de $O(n * \log n)$ ya que es un bucle que recorre todas las implicaciones y en cada iteración recorre las implicaciones aún no recorridas por el bucle principal.

Sin embargo, la aplicación de manera sucesiva del algoritmo para eliminar varias variables no tiene por qué tener una tendencia a tardar menos tiempo ya que al aplicar el operador de olvido sobre dos implicaciones podemos generar varias (0, 1 ó 2) y estamos realizando esta operación de manera combinatoria sobre el conjunto completo. En definitiva, una aplicación de este algoritmo sobre un conjunto reduce en uno (o en cero si el conjunto no contenía previamente la variable) la cantidad de atributos diferentes pero puede aumentar la cantidad de implicaciones contenidas por el conjunto.

Optimizaciones sobre el algoritmo

EL aumento del número de implicaciones mencionado en el apartado anterior es el factor que más afecta al rendimiento de ejecuciones sucesivas del algoritmo por lo que se han aplicado dos optimizaciones para intentar reducir sus efectos:

Implicaciones con consecuete vacío Una implicación con el consecuente (la parte a la derecha de la flecha) vacío no es útil a la hora de realizar razonamiento sobre un conjunto ya que no aporta ninguna información nueva. Por este motivo,

todas las implicaciones que cumplan esta característica se filtran del conjunto final de implicaciones.

Implicaciones con el antecedente vacío Cuando una implicación no contiene ningún atributo a la izquierda (antecedente) significa que todos los atributos del consecuente pueden considerarse como ciertos sin ninguna precondición.

Basándonos en esto podemos sustituir todas la apariciones de esos atributos por el valor lógico *cierto*. Y podemos ir más allá, puesto que tanto antecedente como consecuente de las implicaciones son conjunciones podemos aplicar que:

$$\mathrm{T} \wedge p \rightarrow p$$

Lo cual nos permite directamente eliminar todas las ocurrencias del atributo en cuestión en lugar de simplemente sustituirlas.

Esta optimización no produce de forma directa una reducción del número de implicaciones, pero su aplicación hace más probable que ocurran casos que permitan aplicar la optimización anterior al reducirse el número de atributos.

Estas optimizaciones evidentemente no reducen el orden de complejidad del algoritmo pero reducen el número de reglas del conjunto, provocando de manera indirecta una reducción del tiempo cómputo de ejecuciones sucesivas del mismo.

Programa ejecutable

Con el objetivo de facilitar el uso del algoritmo su distribución se realiza empaquetándolo en un archivo ejecutable Java (.jar) de forma que puede ser utilizado en todo sistema compatible con Java y llamado desde cualquier lenguaje de programación o de forma manual desde un terminal de línea de comando.

El ejecutable lee las implicaciones de un archivo de texto indicado por parámetro con el siguiente formato:

- Una implicación por línea
- Atributos separados por comas.
- Antecedente y consecuente de una implicación separados por el símbolo =>.
- No es necesario respetar ningún tipo de espaciado entre elementos.

Por ejemplo:

```
n  =>  c, d
a, b, c, p  =>  g, r, t
d  =>  c
g  =>  c
r  =>  p, t, a,  b, g, c
a  =>  b, p
p, t  , a, d, b, g, c, r  =>  n
t  =>  p, a, b, g, c, r
b, c, g  =>  a, p, r, t
```

El programa realiza una comprobación sobre la estructura de este archivo y es capaz de indicar el lugar en el que se ha cometido un error.

A la hora de ejecutar el algoritmo, la aplicación permite algunas opciones:

Formato Otter Permite la opción de mostrar el resultado de la ejecución en formato Otter además del formato análogo al del fichero de entrada.

Traza El programa permite mostrar una traza de la ejecución del algoritmo en la que se muestra el proceso de eliminación de cada una de las variables elegidas así como la procedencia de cada implicación (indicando las implicaciones sobre las que se aplicó el operador de olvido para obtener la nueva implicación).

Tiempo de ejecución Permite la opción de mostrar el tiempo de ejecución del algoritmo tras terminar (sin contar el tiempo utilizado para el parseo de archivo u otras operaciones).

Sin optimizaciones Permite ejecutar el algoritmo sin las optimizaciones explicadas en el apartado anterior.

Como se referencia esto?

Para facilitar la programación de la utilidad de línea de comando se ha utilizado una librería llamada *scopt* que facilita la creación de un parser de opciones de línea de comando al estilo típico de Linux.

Esta librería permite declarar de forma simple las opciones y su efecto y genera automáticamente documentación para el uso de la aplicación de línea de comando. La salida de la aplicación al cometer un error al llamarla o al utilizar la opción *-help* es:

```
Implication Retractor 2.1
Usage: Implication Retractor [options] <file>

<file>              Fichero que contiene las fórmulas
-v, --vars <value>  Lista de variables a eliminar
-o, --otter         Mostrar salida en formato Otter
-t, --trace         Mostrar traza de la ejecución
-T, --timed         Mostrar el tiempo de ejecución del algoritmo
--version1          Ultizar versión básica del algoritmo sin optimizaciones
```

Y una llamada de ejemplo al mismo podría ser:

```
retractor.jar implicaciones.txt -v p,q,r --version1
```

La cual generaría una retracción conservativa de las implicaciones contenidas en *implicaciones.txt* sin los atributos *p, q* y *r* aplicando el algoritmo sin optimizaciones.

Pruebas y resultados

Para analizar el funcionamiento del algoritmo así como de las optimizaciones se han realizado ejecuciones de ambas versiones sobre diferentes conjuntos de reglas generados mediante FCA a partir de contextos aleatorios de diferentes tamaños.

Sobre cada conjunto de reglas se ha ejecutado el algoritmo hasta eliminar todas las variables presentes en el mismo y se ha monitorizado el tiempo de ejecución y el número de reglas resultante en cada iteración.

Para la generación de reglas se han utilizado contextos de tamaño cuadrado (mismo número de atributos que de ejemplos) que se han rellenado de forma aleatoria en base a un valor de densidad. Esta densidad representa la probabilidad de que un ejemplo posea un atributo.

Resultados generales

En primer lugar analizaremos la ejecución global del algoritmo, es decir la eliminación de todas las variables.

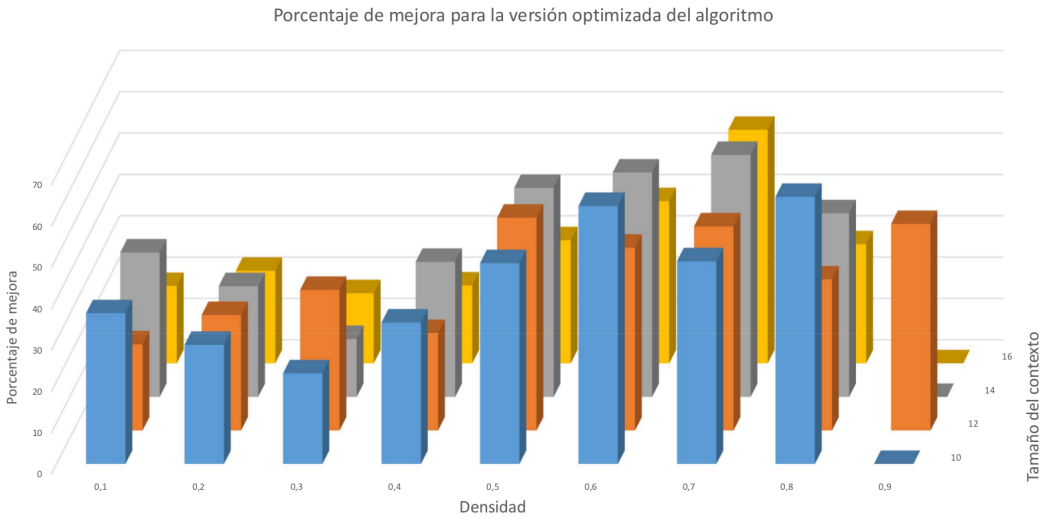

Figura 2: Porcentaje de mejora del algoritmo optimizado sobre el no optimizado en función de densidad y tamaño del contexto inicial.

En la figura 2 (que representa la tabla 2) puede verse información de ejecuciones sobre diferentes conjuntos y el cálculo del porcentaje de mejora de la versión optimizada del algoritmo sobre la no optimizada.

En todos los casos el algoritmo con optimizaciones obtiene mejores resultados (o iguales en casos muy pequeños) que el no optimizado, variando el porcentaje de mejora **entre un 14 % y un 65 % con un valor medio de 36,5 %.**

También puede observarse un mayor peso de la optimización en densidades medias-altas. Uno de los motivos a los que puede deberse esto es que a esas densidades el número resultante de reglas suele ser mayor, lo cual facilita que se creen la situaciones que las optimizaciones pueden tratar.

Cuadro 2: Tiempos de ejecución de las dos versiones del algoritmo (v1: sin optimizaciones, v2: optimizado) para diferentes conjuntos de reglas

Tamaño Cxto	Densidad	Nº reglas	T v1 (ms)	T v2 (ms)	% mejora
10	0,1	15	104	66	36,5
10	0,2	22	135	96	28,9
10	0,3	14	50	39	22,0
10	0,4	24	452	297	34,3
10	0,5	17	37	19	48,6
10	0,6	15	8	3	62,5
10	0,7	20	14	11	21,4
10	0,8	15	17	6	64,7
10	0,9	3	0	0	0
12	0,1	17	206	163	20,9
12	0,2	23	1015	731	28,0
12	0,3	31	734	484	34,1
12	0,4	30	89	68	23,6
12	0,5	49	1735	841	51,5
12	0,6	47	1130	630	44,2
12	0,7	31	170	86	49,4
12	0,8	35	82	52	36,6
12	0,9	7	2	1	50,0
14	0,1	41	16222	10549	35,0
14	0,2	36	9315	6818	26,8
14	0,3	77	682889	587106	14,0
14	0,4	81	121909	82034	32,7
14	0,5	56	3714	1835	50,6
14	0,6	68	6632	3030	54,3
14	0,7	95	9347	3873	58,6
14	0,8	39	135	75	44,4
14	0,9	7	0	0	0
16	0,1	41	164563	133748	18,7
16	0,2	53	84052	65228	22,4
16	0,3	84	13513731	11221599	17,0
16	0,4	110	3370540	2735275	18,8
16	0,5	101	209397	147008	29,8
16	0,6	148	1734667	1053872	39,2
16	0,7	80	3101	1349	56,5
16	0,8	53	243	173	28,8
16	0,9	6	0	2	0

Análisis por iteración

Ya que el objetivo del las optimizaciones implementadas es reducir el tamaño del conjunto en ejecuciones sucesivas del algoritmo, se han analizado las ejecuciones de la tabla 2 iteración por iteración para poder ver el efecto concreto de dichas optimizaciones sobre los conjuntos de reglas.

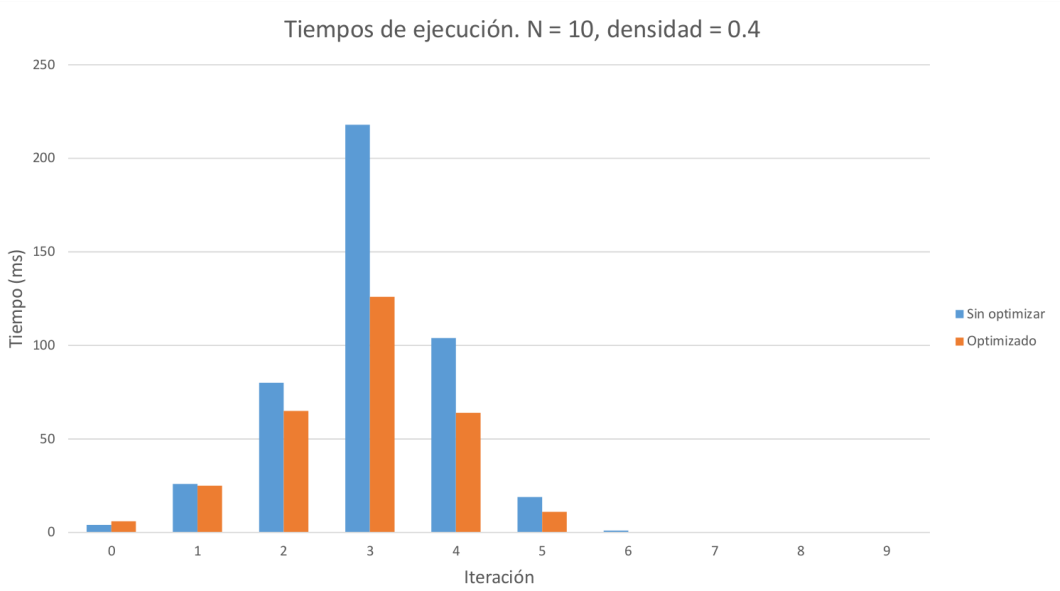

Figura 3: Tiempos de ejecución por iteración para un contexto de tamaño 10 y densidad 0.4

En las figuras 3, 4, 5 se muestran algunas de estas ejecuciones en detalle.

En general, puede verse que la primera iteración presenta el mismo tiempo de ejecución para ambas versiones o en ocasiones un tiempo ligeramente mayor para la versión optimizada. Esto es lógico ya que la versión optimizada realiza más operaciones sobre el conjunto pero no se beneficia de las optimizaciones.

A partir de la segunda iteración, el número de reglas del conjunto es siempre inferior en la versión optimizada, lo cual repercute en una mejora sobre los tiempos de ejecución.

Estas trazas también permiten apreciar que en general el algoritmo provoca un crecimiento en el número de reglas con cada eliminación hasta que llega a un punto de inflexión tras el cual el número de reglas disminuye tras cada ejecución.

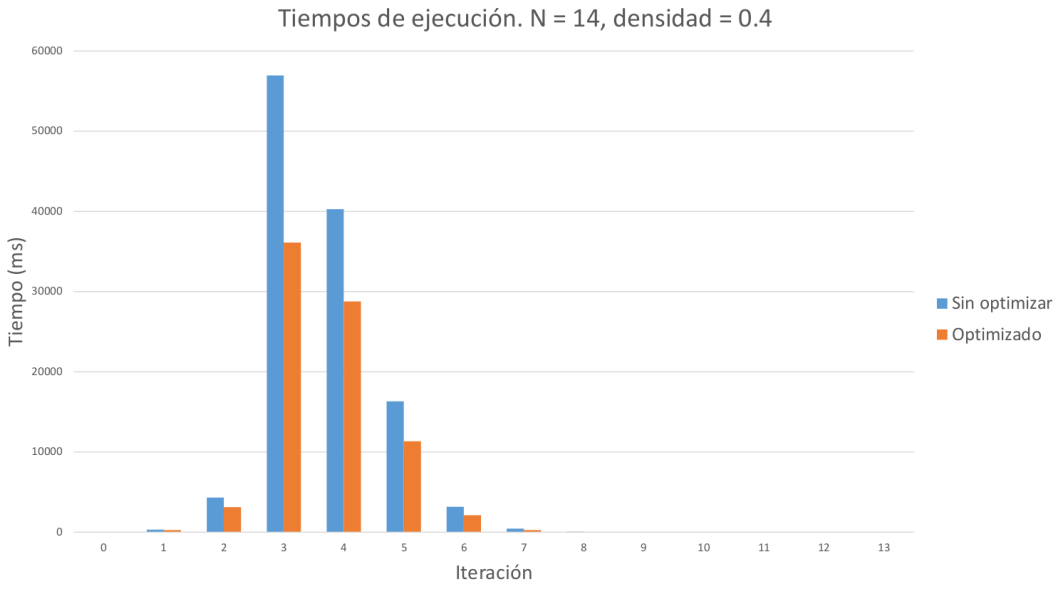

Figura 4: Tiempos de ejecución por iteración para un contexto de tamaño 14 y densidad 0.4

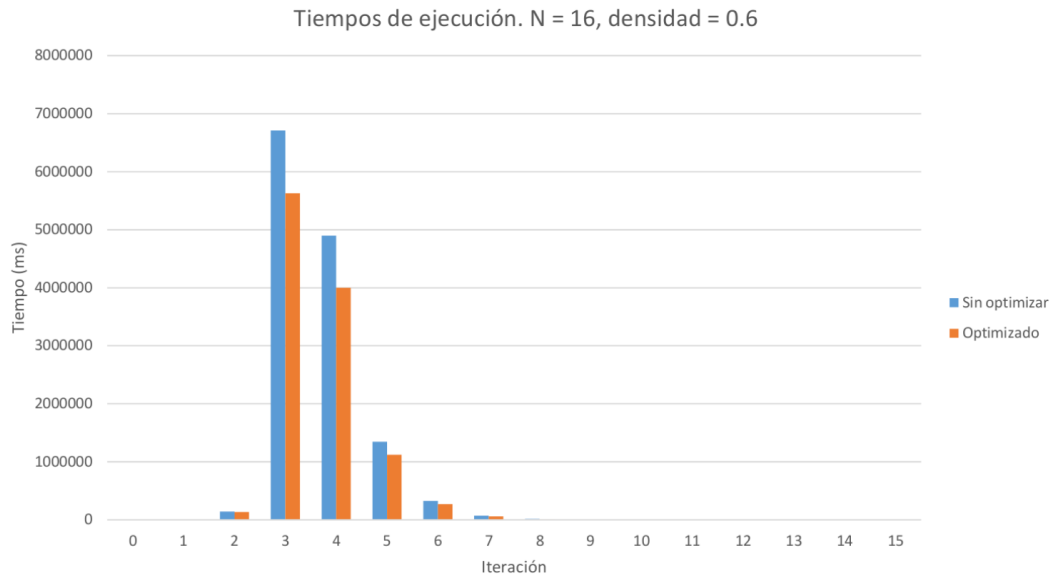

Figura 5: Tiempos de ejecución por iteración para un contexto de tamaño 16 y densidad 0.3

Cuadro 3: Detalle de las iteraciones con un contexto 10x10 y densidad 0.4

Iteración	T v1 (ms)	T v2 (ms)	Tamaño Salida v1	Tamaño Salida v2
0	4	6	58	54
1	26	25	104	91
2	80	65	186	132
3	218	126	139	103
4	104	64	57	39
5	19	11	15	7
6	1	0	3	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0

Cuadro 4: Detalle de las iteraciones con un contexto 14x14 y densidad 0.4

Iteración	T v1 (ms)	T v2 (ms)	Tamaño Salida v1	Tamaño Salida v2
0	33	33	263	233
1	306	258	977	798
2	4321	3130	3858	2908
3	56965	36119	3178	2576
4	40276	28759	2092	1673
5	16315	11330	922	727
6	3171	2108	349	252
7	457	267	118	74
8	59	28	38	18
9	6	2	11	4
10	1	0	4	0
11	0	0	1	0
12	0	0	0	0
13	0	0	0	0

Cuadro 5: Detalle de las iteraciones con un contexto 16x16 y densidad 0.3

Iteración	T v1 (ms)	T v2 (ms)	Tamaño Salida v1	Tamaño Salida v2
0	38	40	520	509
1	1324	1360	5611	5279
2	142854	135685	33640	29701
3	6710792	5628368	29159	25214
4	4896754	3996734	15526	13530
5	1346728	1119486	7472	6487
6	325921	268265	3625	3132
7	72416	58667	1577	1339
8	14593	11322	642	525
9	2311	1672	239	182
10	333	220	84	57
11	39	21	23	11
12	3	1	6	1
13	0	0	2	0
14	0	0	0	0
15	1	0	0	0

Parte III

Conclusiones

Conclusiones, limitaciones y trabajo futuro

La implementación realizada del retractor de implicaciones nos proporciona una prueba de concepto de que esta teoría puede aplicarse de forma satisfactoria a FCA (o a cualquier otro sistema que trabaje con implicaciones o proposiciones) y puede resultar útil a la hora de reducir la complejidad de los conjuntos o a eliminar variables que no nos interesan.

Sin embargo nuestro programa esta implementado de forma simple y a pesar de las optimizaciones realizadas no es eficiente a la hora de tratar con grandes cantidades de datos. Esto se debe a la naturaleza combinatoria del problema que añade una gran complejidad computacional.

Uno de los posibles trabajos futuros que derivan de forma inmediata, es realizar una implementación basada en computación distribuida (por ejemplo utilizando Apache Spark) que nos permita tratar de forma paralela el bucle principal del algoritmo en el que se realizan las combinaciones. Ya que cada iteración no depende de las anteriores es sencillo realizar una computación distribuida y finalmente unir los resultados.

Por último, otra posibilidad de trabajo futuro es la refactorización de la librería de lógica junto con la construcción de una API pública para la misma, que permita su distribución como librería genérica de programación lógica en el entorno Java facilitando la creación de algoritmos y programas que trabajen con este tipo de datos.

Parte IV

Bibliografía

Bibliografía

[1] Joaquín Borrego-Díaz Gonzalo A. Aranda-Corral and Juan Galán-Páez. Conservative retractions of implication bases.

[2] M. Magdalena Fernández-Lebrón Gonzalo A. Aranda-Corral, Joaquín Borrego-Díaz. Conservative retractions of propositional logic theories by means of boolean derivatives. theoretical foundations.

[3] José A. Alonso Jiménez. *Lógica en Haskell*. Dpto. de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla, Sevilla, 2008.

[4] Gonzalo A. Aranda-Corral y Joaquín Borrego-Díaz José A. Alonso-Jiménez. Sistema certificado de decisión proposicional basado en polinomios. 2009.

lugares de publicacion? fechas? Referencias a la tesis FCA?