Desarrollo de un Prototipo de software para probar condiciones que deben cumplir Funciones Lógicas Cuaternarias Regulares

Cristian Camilo Mora Sabogal

Universidad Distrital Francisco José de Caldas
Facultad de Ingeniería
Proyecto Curricular de Ingeniería de Sistemas
Bogotá D.C., Colombia
2020

Desarrollo de un Prototipo de software para probar condiciones que deben cumplir Funciones Lógicas Cuaternarias Regulares

Cristian Camilo Mora Sabogal

Trabajo de grado para optar por el título de
Ingeniero de Sistemas

Dirigido por:
Ing. Msc. Jose Jairo Soriano Mendez

Universidad Distrital Francisco José de Caldas
Facultad de Ingeniería
Proyecto Curricular de Ingeniería de Sistemas
Bogotá D.C., Colombia
2020

Índice general

1. DESCRIPCIÓN DEL PROYECTO **3**
- 1.1. Resumen . . . 3
- 1.2. Planteamiento del problema . . . 3
- 1.3. Justificación . . . 4

2. OBJETIVOS **5**
- 2.1. Objetivo general . . . 5
- 2.2. Objetivos específicos . . . 5

3. MARCO DE REFERENCIA **6**
- 3.1. Proceso Unificado (PU) . . . 6
- 3.2. Desarrollo Ágil . . . 7
- 3.3. Proceso Unificado Ágil (PUA) . . . 7
- 3.4. Álgebra de De Morgan . . . 8
- 3.5. Formación de Lógicas Proposicionales . . . 9
- 3.6. Funciones Lógicas Cuaternarias Regulares (FLCR) . . . 10
- 3.7. Obteniendo FLCR a partir de tablas de verdad . . . 10

4. ESTADO DEL ARTE **12**

5. METODOLOGÍA **13**
- 5.1. Hipótesis . . . 13
- 5.2. Tipo de estudio . . . 13
- 5.3. Unidad de análisis . . . 13
- 5.4. Unidad de estudio . . . 13
- 5.5. Metodología de Ingeniería . . . 14

6. DESARROLLO DE REQUERIMIENTOS **15**
- 6.1. Requerimientos . . . 15
- 6.2. Casos de uso . . . 16

7. DEFINICIÓN DE LA ARQUITECTURA **20**

8. MODELAMIENTO DEL SISTEMA **22**
- 8.1. Diagramas de actividades . . . 22
  - 8.1.1. Generar tablas predefinidas . . . 22
  - 8.1.2. Generar tablas parciales . . . 24

8.1.3. Convertir tablas a función . . . 26
8.1.4. Convertir función a tabla . . . 27
8.1.5. Verificación T-T . . . 27
8.2. Diagrama de componentes . . . 29

9. CONDICIONES PARA FILTRADO DE TABLAS 30
9.1. Condiciones referenciadas . . . 30
9.1.1. Condición 1: Fórmulas bien formadas . . . 30
9.1.2. Condición 2: Regularidad para la incertidumbre . . . 30
9.1.3. Condición 3: Monotonicidad para la incertidumbre . . . 31
9.1.4. Condiciones 4 y 5: Incertidumbre combinada . . . 31
9.2. Condiciones propuestas . . . 32
9.2.1. Condición 1: Regularidad . . . 32
9.2.2. Condición 2: Reducibilidad . . . 34
9.2.3. Condición 3: Inversión . . . 35
9.2.4. Condición 4: Repetición . . . 36
9.2.5. Condición 5: Restricción . . . 36
9.2.6. Ejemplo . . . 37

10. RESULTADOS 38
10.1. Condiciones referenciadas . . . 38
10.1.1. Generar Tablas predefinidas . . . 38
10.1.2. Verificación T-T . . . 38
10.2. Condiciones propuestas . . . 38
10.2.1. Tablas predefinidas . . . 39
10.2.2. Tablas parciales . . . 39
10.2.3. Verificación T-T . . . 40

11. CONCLUSIONES 41

12. TRABAJO FUTURO 42

13. ANEXO 43
13.1. Relaciones . . . 43
13.2. Retículos . . . 44

Capítulo 1

DESCRIPCIÓN DEL PROYECTO

1.1. Resumen

El documento desarrollado consiste en una propuesta de un sistema de software que permita probar condiciones necesarias y suficientes que deben cumplir tablas de verdad representadas por funciones lógicas en el álgebra de De Morgan, a partir de una metodología de desarrollo de software, y la evaluación del sistema con condiciones encontradas en la literatura, y condiciones propuestas.

1.2. Planteamiento del problema

Las funciones lógicas han sido usadas en varias aplicaciones de ingeniería para resolver problemas no lineales. Las funciones lógicas basadas en lógica binaria (verdadero o falso) son extensamente aplicadas en circuitos, algoritmos y controladores [1, 2], sin embargo esos son problemas en donde el uso de lógica binaria resulta inadecuado por su estructura [3, 4]. Así, cuando se modela los estados transitorios de un circuito o se agrega suavidad a los controladores, se prefieren las funciones lógicas ternarias (para simplificar sistemas difusos) [5], y en las simplificaciones de los sistemas difusos de intervalo o sistemas difusos de tipo 2, son usadas funciones lógicas cuaternarias para resolver problemas no lineales [6, 7].

Cuando se aplican las funciones ternarias o cuaternarias rara vez se utilizan todas las funciones resultantes, en su lugar sólo son necesarios algunos subconjuntos de funciones como las funciones con propiedades o significados especiales. De acuerdo con lo anterior, Mukaidono estableció tres condiciones necesarias y suficientes para que las tablas generadas en la lógica ternaria de n variables puedan ser representadas por funciones lógicas, las funciones que satisfacen las condiciones son llamadas funciones lógicas ternarias regulares [8]. Gehrke, Carol Walker y Elbert Walker [10, 11] encontraron un algoritmo para obtener la forma normal disyuntiva de las funciones lógicas cuaternarias, en [11] un homomorfismo fue realizado de las funciones lógicas cuaternarias a las funciones lógicas booleanas, encontrando que el número de funciones cuaternarias que representan las funciones booleanas o el número de tablas que pueden ser representadas por funciones lógicas cuaternarias es 168 para el caso de dos variables, un número que coincide con el cuarto número de Dedeking, el homomorfismo puede ser extendido para n variables coincidiendo con el $2n$ número de Dedeking utilizando las cardinalidades computadas

en [12] para 1, 2, y 3 variables, las funciones cuaternarias que representan tablas son llamadas funciones lógicas cuaternarias regulares.

Mas recientemente, Soriano en [13] propuso 5 condiciones para que el homomorfismo de n variables se cumpla, pero si se quiere probar las tablas que representen las funciones lógicas cuaternarias de todas las tablas posibles, existe una desigualdad $2^{2^n} \leq 2n$ #Dedeking $\leq 4^{4^n}$, donde 2^{2^n} es el número de tablas de Boole, y 4^{4^n} es el número de tablas de De Morgan, el cual es muy grande en $n \geq 2$ para probar todas las tablas a la vez o iterativamente [11].

De acuerdo con lo anterior ¿Cuales son los procedimientos o métodos relevantes a la hora de probar todas las tablas que puedan ser representadas por funciones lógicas cuaternarias?, ¿Cuales relaciones existentes en la formación de Formas Normales para Funciones Lógicas Cuaternarias Regulares permitirían descartar de antemano una gran cantidad de tablas antes de ser probadas?

1.3. Justificación

En los trabajos encontrados acerca de algoritmos en los que se usa álgebras de Kleene y de De Morgan [7, 12, 19, 20, 21, 22, 23], no se ha encontrado alguno en la que implemente un software y trabaje con una metodología de desarrollo de software, fomentar estos cambios puede ayudar en la transparencia y el entendimiento no solamente del algoritmo, también en el ofrecimiento de un producto de software y su correspondiente proceso de desarrollo (requerimientos, análisis, diseño, implementación, y pruebas).

El software propuesto puede ser un punto de partida para la comparación y descarte de grandes cantidades de datos jerarquizados a la hora de probar condiciones en distintas lógicas proposicionales, también puede ayudar al cálculo de cardinalidades.

Además se pretende que pueda ayudar en el área de automatización, ya que se ha trabajado en la lógica de De Morgan, la cual simplifica la lógica difusa de intervalo usada en controladores con varios observadores u otras áreas donde existan problemas no lineales, a una lógica tetravalente, donde se pueda usar o elegir Funciones Lógicas Cuaternarias, y se requiera descartar información no relevante para averiguar si las funciones a trabajar son o no Funciones Lógicas Cuaternarias Regulares.

Capítulo 2

OBJETIVOS

2.1. Objetivo general

Implementar un prototipo de software para probar condiciones necesarias y suficientes que deben cumplir las tablas generadas en el Álgebra de De Morgan de 1, 2, y 3 variables representadas por funciones lógicas.

2.2. Objetivos específicos

- Determinar el diseño del prototipo del software por medio de una arquitectura centrada en el flujo de datos.
- Establecer una metodología partiendo del conocimiento acerca de las Funciones Lógicas Cuaternarias Regulares que permitan descartar una gran cantidad de información antes de la prueba de las condiciones.
- Evaluar la eficiencia del software, utilizando una metodología de desarrollo de software.
- Evaluar la eficacia del software, probando las condiciones propuestas por Soriano en [13], y las condiciones propuestas en este trabajo.

Capítulo 3

MARCO DE REFERENCIA

Para fundamentar el proceso de conocimiento es necesario describir tanto una metodología de software de confianza, como el desarrollo para llegar a las Funciones Lógicas Cuaternarias Regulares, presentados a continuación.

3.1. Proceso Unificado (PU)

Es un proceso de desarrollo de software que hace énfasis en la importancia de la arquitectura del software y "ayuda a que el arquitecto se centre en las metas correctas, tales como que sea comprensible, permita cambios futuros y la reutilización" planteando un flujo del proceso iterativo e incremental, lo que da la sensación evolutiva que resulta esencial en el desarrollo moderno del software [14], el proceso unificado consta de 4 fases, que son las siguientes:

La fase de concepción o gestación agrupa actividades tanto de la toma de requerimientos con el cliente como del análisis de los mismos. Al colaborar con los participantes, se propone una arquitectura aproximada para el sistema y se desarrolla un plan para la naturaleza iterativa e incremental del proyecto en cuestión.

La fase de elaboración incluye las actividades de análisis de requerimientos y diseño del modelo general del software. La elaboración mejora y amplía los casos de uso preliminares desarrollados como parte de la fase de concepción y aumenta la representación de la arquitectura.

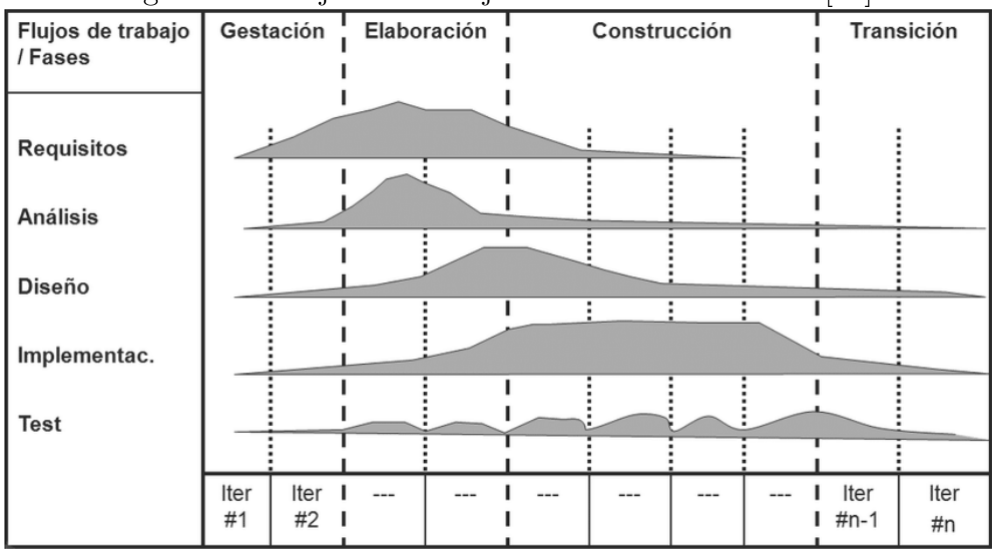
La fase de construcción es idéntica a la actividad de implementación definida para el proceso general del software. Con el uso del modelo de arquitectura como entrada, la fase de construcción desarrolla o adquiere los componentes del software que harán que cada caso de uso sea operativo para los usuarios finales.

La fase de transición incluye las últimas etapas de la actividad general de construcción, las pruebas y la primera parte de la actividad de despliegue general (entrega y retroalimentación). Se da el software a los usuarios finales para las pruebas beta, quienes reportan tanto los defectos como los cambios necesarios.

Aunque se nombren las actividades que más utilizan en el proceso general de software (Requerimientos, Análisis, Diseño, Implementación, Prueba), cada fase del proceso unificado

toma todas las actividades aunque con intensidad diferente como se presenta en la Figura 3.1.

Figura 3.1: Flujos de trabajo del Proceso Unificado [15].



3.2. Desarrollo Ágil

Cualquier proceso del software ágil se caracteriza por la forma en la que aborda cierto número de suposiciones clave acerca de la mayoría de proyectos de software [14, 16]:

1. Es difícil predecir qué requerimientos de software persistirán y cuáles cambiarán. También es difícil pronosticar cómo cambiarán las prioridades del cliente a medida que avanza el proyecto.

2. Para muchos tipos de software, el diseño y la construcción están implicados. Es decir, ambas actividades deben ejecutarse en forma simultánea, de modo que los modelos de diseño se prueben a medida que se crean. Es difícil predecir cuánto diseño se necesita antes de que se use la construcción para probar el diseño.

3. El análisis, el diseño, la construcción y las pruebas no son tan predecibles como nos gustaría (desde un punto de vista de planeación).

3.3. Proceso Unificado Ágil (PUA)

Adopta una filosofía “en serie para lo grande” e “iterativa para lo pequeño” a fin de construir sistemas basados en computadora. Al adoptar las actividades en fase clásicas del PU (concepción, elaboración, construcción y transición), el PUA brinda un revestimiento en serie (por ejemplo, una secuencia lineal de actividades de ingeniería de software) que permite visualizar el flujo general del proceso de un proyecto de software. Sin embargo, dentro de cada actividad, se repite con objeto de alcanzar la agilidad y entregar tan rápido como sea posible incrementos de software significativos a los usuarios finales [14, 16]. Cada iteración del PUA aborda las actividades siguientes:

- Modelado. Se crean representaciones de UML de los dominios del negocio y el problema. No obstante, para conservar la agilidad, estos modelos deben ser "sólo suficientemente buenos" para permitir el avance.
- Implementación. Los modelos se traducen a código fuente.
- Pruebas. Se diseña y ejecuta una serie de pruebas para detectar errores y garantizar que el código fuente cumple sus requerimientos.
- Despliegue. En este contexto se centra en la entrega de un incremento de software y en la obtención de retroalimentación de los usuarios finales.
- Configuración y administración del proyecto. La administración de la configuración incluye la administración del cambio y el riesgo, y el control de cualesquiera productos del trabajo persistentes que se produzcan.

3.4. Álgebra de De Morgan

Para el desarrollo del software se debe conocer cómo se forman las Funciones Lógicas Cuaternarias Regulares, un buen punto de inicio es la definición de el álgebra de De Morgan y la construcción de una álgebra de De Morgan tetravalente, las cuales se verán en esta sección.

Definición 1 *(Álgebra de De Morgan) Un álgebra* $(L, \wedge, \vee, ', 0, 1)$*, con operaciones binarias de conjunción* $(\wedge)$*, disyunción* $(\vee)$*, la operación unaria de complemento* $(')$*, y las operaciones nularias o constantes* 0 *y* 1 *es un álgebra de De Morgan si* $(L, \wedge, \vee)$ *es un retículo distributivo,* $(L, \wedge, \vee, 0, 1)$ *es un retículo acotado, y se satisfacen las siguientes propiedades para todos los* $a, b, c \in L$ [10]:

$$\text{(Leyes de De Morgan) } (a \vee b)' = a' \wedge b';\ (a \wedge b)' = a' \vee b' \tag{3.1}$$

$$\text{(Involución) } (a')' = a \tag{3.2}$$

Definición 2 *(Álgebra de De Morgan tetravalente) Es un álgebra de De Morgan* $\mathbb{D} = (D, \wedge, \vee, ', 0, 1)$ *donde el conjunto* D *es el retículo de la* Figura 3.2*. Las operaciones binarias, y la operación unaria son definidas en la* Tabla 3.1 [10, 11]*:*

Figura 3.2: Diagrama del retículo del álgebra de De Morgan tetravalente D. [11].

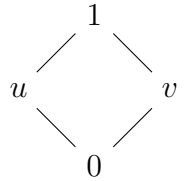


Tabla 3.1: Operaciones del álgebra de De Morgan tetravalente. (a) Conjunción, (b) Disyunción, (c) Complemento.

$\wedge$	0	u	v	1
0	0	0	0	0
u	0	u	0	u
v	0	0	v	v
1	0	u	v	1

(a)

$\vee$	0	u	v	1
0	0	u	v	1
u	u	u	1	1
v	v	1	v	1
1	1	1	1	1

(b)

	$'$
0	1
u	u
v	v
1	0

(c)

3.5. Formación de Lógicas Proposicionales

En esta sección se continua con la definición de fórmulas bien formadas, la construcción de las álgebras de De Morgan a partir de las fórmulas, y la construcción de una lógica proposicional, también es denotada la diferencia entre un álgebra y una lógica proposicional.

Definición 3 *(Conjunto de fórmulas bien formadas) Sea V un conjunto finito de variables $x_1, x_2, ..., x_i, ..., x_n$ no vacío. Dadas las constantes 0 y 1, el conjunto V y los símbolos de operación $\wedge$, $\vee$, $'$, se define el conjunto T de la siguiente manera:*

1. *$0 \in T$, $1 \in T$, y si $x_i \in V$ entonces $x_i \in T$.*
2. *Si $f \in T$ y $g \in T$ entonces $f \wedge g \in T$, $f \vee g \in T$, $f' \in T$.*

La definición anterior es una manera de evitar concatenaciones sin sentido de símbolos como $x_1x_2 \vee x_3\wedge$ que no se considera una fórmula bien formada. Si estos símbolos se concatenan de acuerdo con la definición puede obtenerse por ejemplo $x_1 \vee (x_2 \wedge x_3)$ [9, 10, 11].

Definición 4 *(Álgebra de fórmulas) Es un álgebra de De Morgan $\mathbb{T} = (T, \wedge, \vee, ', 0, 1)$ donde el conjunto T es un conjunto de fórmulas bien formadas, y si $f, g \in T$ entonces $f \wedge g \in T$, $f \vee g \in T$, $f' \in T$.*

Dos términos $f, g \in T$ son equivalentes en $\mathbb{D}$ si y solo si $f(a_1, a_2, ..., a_n) = g(a_1, a_2, ..., a_n)$ para todo $a_1, a_2, ..., a_n \in D$. Esta relación de equivalencia es una congruencia que permite las clases de equivalencia en si mismas, de modo que son combinadas utilizando los conectivos dados [9, 10].

Definición 5 *(Lógica proposicional) Es un álgebra de De Morgan $\mathbb{L}_\mathbb{D} = (L_D, \wedge, \vee, ', 0, 1)$, también llamada lógica proposicional en las variables del conjunto V con valores del álgebra $\mathbb{D}$, donde L_D es un retículo de clases de equivalencia de los términos en el álgebra $\mathbb{T}$, y si $[f], [g] \in L_D$, entonces $[f \wedge g] \in L_D$, $[f \vee g] \in L_D$, $[f'] \in L_D$ [9, 10, 11].*

3.6. Funciones Lógicas Cuaternarias Regulares (FLCR)

Una forma normal es un elemento representativo de las clases de equivalencia en una lógica proposicional. Específicamente para $\mathbb{L}_{\mathbb{D}}$ las formas normales serán llamadas Funciones Lógicas Cuaternarias Regulares. En esta sección son nombradas los fundamentos para obtener la forma normal disyuntiva en $\mathbb{L}_{\mathbb{D}}$.

Definición 6 *($\vee$-irreducible) Un elemento f de un retículo finito $(L, \wedge, \vee)$ es $\vee$-irreducible, si $f \neq 0$, y si $f = g \vee h$, entonces $f = g$ o $f = h$, $f, g, h \in L$ [12].*

Proposición 1 *En un retículo distributivo finito, cada elemento es la unicamente la disyunción de los $\vee$-irreducibles incomparables.*

Proposición 2 *Cada elemento en $\mathbb{L}_{\mathbb{D}}$ es unicamente la disyunción de los $\vee$-irreducibles incomparables y cada $\vee$-irreducible es la conjunción de los literales (variables y sus negaciones).*

Teorema 1 *Sean a, b conjunciones de diferentes literales en $\mathbb{L}_{\mathbb{D}}$, con $b \leq a$, entonces:*

- *Cada literal que existen en a, existe en b.*
- *$a = b$ si y solo si son conjunciones de exactamente los mismos literales.*
- *a es $\vee$-irreducible.*

Este teorema dice cuales son los $\vee$-irreducibles, y dice que hay un orden entre ellos.

3.7. Obteniendo FLCR a partir de tablas de verdad

Las tablas para todos los valores de verdad $a_1, a_2, ..., a_n \in D$, con variables $x_1, x_2, ..., x_n$, $f(x_1, x_2, ..., x_n) = t \in V$ son llamadas tablas de verdad, donde f no es necesariamente una fórmula bien formada, en esta sección se nombra el procecimiento para obtener formas normales disyuntivas en $\mathbb{L}_{\mathbb{D}}$ a partir de tablas de verdad, los $\vee$-irreducibles básicos están en la Tabla 3.2 [10, 11]:

Tabla 3.2: Obteniendo $\vee$-irreducibles de las tablas de verdad.

x_1	x_2	x_3	x_4	t	$\vee$-irreducible
1	u	0	0	1	$x_1 \wedge x_3' \wedge x_4'$
1	v	0	u	1	$x_1 \wedge x_2 \wedge x_2' \wedge x_3'$ y $x_1 \wedge x_3' \wedge x_4 \wedge x_4'$
0	u	1	v	u	$x_1' \wedge x_2 \wedge x_2' \wedge x_3$

- Para las filas que tienen valor 1 en la columna de la expresión t, con valores de verdad 0 or 1 y posiblemente u o v (pero no ambos), formar la conjunción con las variables de los valores de verdad iguales a 1, con la negación de las variables de los valores de verdad iguales a 0 (y dejar de lado las variables que tengan valores de verdad iguales a u o v). Si la fila tiene todos los valores de verdad iguales a u (o todos v), el $\vee$-irreducible es igual a 1.

- Para las filas que tienen valor 1 en la columna de la expresión t, con valores de verdad u y v además de los valores de verdad 0 o 1, formar dos conjunciones: la primera conjunción usando las variables de los valores de verdad iguales a 1, la negación de las variables de los valores de verdad iguales a 0, y usando ambos, las variables y la negación de las mismas para los valores de verdad iguales a u, la segunda conjunción usando las variables de los valores de verdad iguales a 1, la negación de las variables para valores de verdad iguales a 0, y usando ambos, las variables y la negación de las mismas para los valores de verdad iguales a v.

- Para las filas que tienen valor u en la columna de la expresión t, formar una conjunción: usando las variables de los valores de verdad iguales a 1, la negación de las variables con valores de verdad iguales a 0, y usando ambas, las variables y su negación para los valores de verdad iguales a u (omita todas las variables con valores de verdad iguales a v). Análogamente hacer lo mismo si la fila tiene un valor v en la columna de la expresión t.

- Ignorar las filas que tienen valor 0 en la columna de la expresión t para ser puesto en la forma normal disyuntiva.

La forma normal disyuntiva irredundante de De Morgan para la expresión es obtenida por hacer la disyunción de los $\vee$-irreducibles observados en la Tabla 3.2, con el 0, y descartar conjunciones redundantes, esto es, cualquier conjunción que contiene los mismos o posiblemente más literales que otras conjunciones.

Capítulo 4

ESTADO DEL ARTE

Considerando el tema a tratar en este proyecto, se han encontrado trabajos en los cuales se usan algoritmos de diversos tipos [7, 12, 19, 20, 21, 22, 23], los cuales pueden servir como punto de partida y se describirán a continuación:

- Usando un método en el documento [7] es posible la simplificación de formas normales en las álgebras de Kleene y de De Morgan, además de su aplicación en un controlador tipo MISO para el nivel de un tanque cilíndrico.
- Es presentado un algoritmo en el documento [12], el cual computa de un conjunto finito parcialmente ordenado, las cardinalidades de los reticulos distributivos a partir del manejo de retículos en los join-irreducibles o $\vee$-irreducibles, son usados como ejemplo las álgebras Kleene, y De Morgan, para 1, 2, y 3 variables.
- En los documentos [19, 20] son extendidas las cardinalidades del documento [12] en las correspondientes álgebras libres de Kleene y de De Morgan para n variables.
- Usando álgebra de Kleene es desarrollado en el artículo [21] un marco general para la formalización de un conjunto bien conocido problemas de análisis de flujo de datos. Este enfoque permite un elegante pero riguroso tratamiento de los problemas de flujo de datos y proporcionar una descripción de estilo libro de cocina.
- El álgebra de Kleene con un algoritmo de pruebas de un sistema equitativo para manipular programas, adicionalmente con una prueba puramente equitativa usando álgebra de Kleene con pruebas y condiciones de conmutatividad es presentado en [22].
- En [23] son propuestos algoritmos para verificar la equivalencia lingüística de autómatas finitos sobre un gran alfabeto. Son utilizados autómatas simbólicos, donde la función de transición se representa de forma compacta usando diagramas de decisión binarios (BDD).

Nótese que en ninguno de los trabajos propuestos se trabaja con una metodología de desarrollo de software ni con una implementación del mismo.

Capítulo 5

METODOLOGÍA

En este capítulo son presentados aspectos de la metodología como la hipótesis, el tipo de estudio, unidad de análisis, unidad de estudio, y la metodología de la ingeniería.

5.1. Hipótesis

A partir de la relación entre las álgebras de De Morgan $\mathbb{L}_{\mathbb{D}}$ y $\mathbb{D}$, en la obtención de las formas normales, y el conocimiento de una metodología de software ágil, se puede probar condiciones necesarias y suficientes para que las tablas en el álgebra de De Morgan puedan ser representados por funciones.

5.2. Tipo de estudio

El tipo de estudio del proyecto es de tipo exploratorio, porque no se ha desarrollado en la literatura métodos de software para comprobar condiciones específicamente para representación de tablas en Funciones Lógicas Cuaternarias, y este trabajo puede servir de fundamento para nuevas investigaciones por otros autores tanto para la creación de software, como para la creación de condiciones.

5.3. Unidad de análisis

La unidad de análisis es la Tabla de verdad en el Álgebra de De Morgan para 1, 2, o 3 variables, el cual su manejo es mostrado en la siguiente sección.

5.4. Unidad de estudio

Cada tabla de verdad del proyecto se maneja dentro de un arreglo, donde los valores de verdad son manejados como posiciones, y los t, son representados por caracteres o números como sigue: 1 para 0, 2 para u, 3 para v, 4 para 1 y 5 como "valor no importa". Estos valores se introdujeron para un correcto manejo de la memoria y procesamiento.

5.5. Metodología de Ingeniería

Para lograr un buen desarrollo del software, es importante utilizar una metodología de desarrollo o modelo de proceso [14], para lo cual se eligió Proceso Unificado Ágil (PUA), debido a que es una metodología ágil, conserva la solidez del Proceso Unificado (PU) y por su filosofía "En serie para lo grande, iterativo para lo pequeño".

Capítulo 6

DESARROLLO DE REQUERIMIENTOS

En el primer paso para desarrollar el proyecto, son definidos los requerimientos del sistema, y los casos de uso con su correspondientes descripción y visualización. Lo anterior es usado como base para empezar el desarrollo del software.

6.1. Requerimientos

En las reuniones dentro de las fases del PUA, es tomado en cuenta que el software se debe centrar en dos aspectos, la creación de tablas de verdad y su correspondiente evaluación sobre la representación de funciones, aquellos aspectos serán tomados como requerimientos, en la Tabla 6.1 se pueden ver los requerimientos encontrados, con sus correspondientes características.

Tabla 6.1: Tabla de requerimientos.

Id	Nombre	Descripción	Prioridad	Funcional
R1	Manejo en la creación de Tablas de verdad	Consiste en que el usuario pueda mantener un control en la creación de tablas de verdad que representen funciones.	Alta	Si
R2	Control de la evaluación de Tablas de verdad	Este requerimiento consiste en que el usuario pueda mantener un control en la evaluación sobre la representación por funciones de las tablas de verdad.	Alta	Si

6.2. Casos de uso

A partir de los requerimientos encontrados son obtenidos los casos de uso, y su interacción con el usuario mostrados en la Figura 6.1. Cada caso de uso es descrito de forma simple y detallada desde la Tabla 6.2 a la Tabla 6.8 con base en el formato de Cockburn [14, 17].

Figura 6.1: Diagrama de casos de uso.

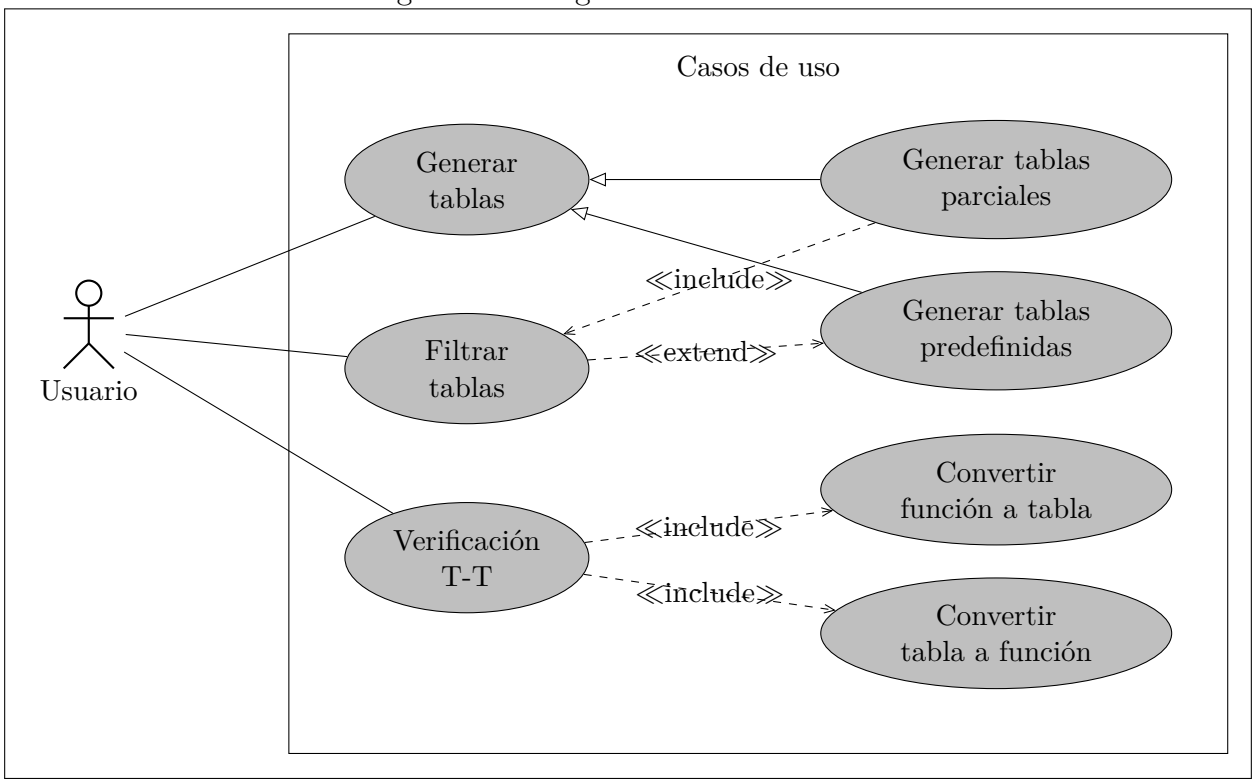


Tabla 6.2: Descripción simplificada caso de uso Generar tablas.

ID	C1
Nombre	Generar Tablas
Descripción: Permite generar todas las tablas de verdad que posteriormente son evaluadas	
Actores: Usuario	

Tabla 6.3: Descripción simplificada caso de uso Filtrar tablas.

ID	C2
Nombre	Filtrar Tablas
Descripción: Filtra las tablas generadas de acuerdo a las condiciones propuestas.	
Actores: Usuario	

Tabla 6.4: Descripción detallada caso de uso Generar tablas predefinidas.

ID	C3
Nombre	Generar tablas predefinidas
Actor	Usuario
Descripción	Genera todas las tablas de verdad a partir de una permutación sin repetición de valores predefinidos
Precondiciones	No aplica
Escenario 1. Usuario: Ingresa a la carpeta donde se encuentra el código 2. Usuario: Ejecuta código	
Excepciones 1. El número de tablas a analizar es muy grande	
Prioridad	Alta
Disponibilidad	6ta iteración
Canal para el actor	Consola o Entorno de desarrollo Integrado (IDE)

Tabla 6.5: Descripción detallada caso de uso Generar tablas parciales.

ID	C4
Nombre	Generar tablas parciales
Actor	Usuario
Descripción	Genera todas las tablas de verdad a partir de la sucesión de permutaciones en tablas incompletas, y filtros por las condiciones.
Precondiciones	Un filtro por condiciones es establecido
Escenario 1. Usuario: Ingresa a la carpeta donde se encuentra el código 2. Usuario: Ejecuta código	
Excepciones 1. El número de tablas a analizar es muy grande	
Prioridad	Media
Disponibilidad	8ta iteración
Canal para el actor	Consola o IDE

Tabla 6.6: Descripción detallada caso de uso Verificación T-T.

ID	C5
Nombre	Verificación T-T
Actor	Usuario
Descripción	Hace las conversiones correspondientes y verifica si la tabla original y la tabla final son iguales.
Precondiciones	Es ingresado un arreglo de valores (1, 2, 3, o 4)
Escenario 1. Usuario: Ingresa a la carpeta donde se encuentra el código 2. Usuario: Ingresa arreglo 3. Usuario: Ejecuta código	
Excepciones 1. No aplica	
Prioridad	Media
Disponibilidad	8ta iteración
Canal para el actor	Consola o IDE

Tabla 6.7: Descripción detallada caso de uso Convertir tabla a función.

ID	C6
Nombre	Convertir Tabla a Función
Actor	Usuario
Descripción	Hace la conversión de una tabla de verdad a una función lógica.
Precondiciones	Es ingresado una tabla de verdad
Escenario No aplica	
Excepciones 1. No aplica	
Prioridad	Alta
Disponibilidad	6ta iteración
Canal para el actor	Consola o IDE

Tabla 6.8: Descripción detallada caso de uso Convertir función a tabla.

ID	C7
Nombre	Convertir Función a Tabla
Actor	Usuario
Descripción	Hace la conversión de una función lógica a una tabla de verdad.
Precondiciones	Es ingresado una función lógica
Escenario No aplica	
Excepciones 1. No aplica	
Prioridad	Alta
Disponibilidad	6ta iteración
Canal para el actor	Consola o IDE

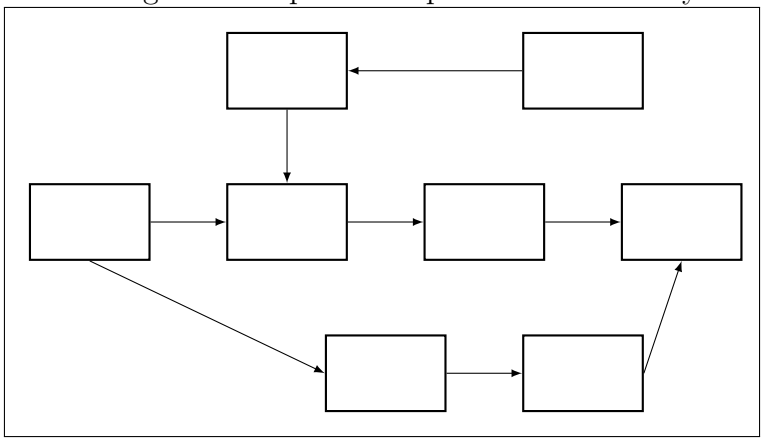
Capítulo 7

DEFINICIÓN DE LA ARQUITECTURA

En el segundo paso para desarrollar el proyecto es definida la arquitectura del sistema, a partir de los requerimientos y casos de uso obtenidos, la opción mas favorable es utilizar el patrón arquitectónico filtros y tuberías descrito a continuación.

El patrón arquitectónico de filtros y tuberías proporciona una estructura para sistemas que procesan un flujo de datos en el que cada paso de procesamiento es encapsulado en un componente de filtro, los datos se pasan a través de tuberías entre filtros adyacentes, y la combinación de filtros le permite construir familias de sistemas relacionados, un ejemplo es mostrado en la Figura 7.1 en el cual los filtros son representados por rectángulos y las tuberías por flechas.

Figura 7.1: Diagrama del patrón arquitectónico filtros y tuberías.



Los componentes del filtro son las unidades de procesamiento de la tubería. Un filtro enriquece, refina o transforma sus datos de entrada. Enriquece los datos por computar y agregar información, refina los datos al concentrar o extrayendo información y transformando datos entregándolos en alguna otra representación. Una implementación de filtro de concreto puede combinar cualquiera de estos tres principios básicos [14, 18]:

- El elemento de tubería posterior extrae datos de salida del filtro.
- El elemento de tubería anterior empuja nuevos datos de entrada al filtro.
- Más comúnmente, el filtro está activo, extrayendo su entrada desde una tubería y empujando su salida por otra tubería.

Capítulo 8

MODELAMIENTO DEL SISTEMA

En este capítulo se muestra el modelamiento del sistema, creado a partir de la definición de la arquitectura, los requerimientos, y los casos de uso obtenidos, donde son utilizados diagramas sólo suficientemente buenos para lograr el avance del software según la metodología de desarrollo de software PUA, específicamente se utilizan los diagramas de actividades, y componentes.

8.1. Diagramas de actividades

La mayoría de casos de uso tienen un diagrama de actividad que permite ver su comportamiento, y un algoritmo que permita profundizarlo, exceptuando Filtrar tablas porque depende de las condiciones, mostradas en el siguiente capítulo, y Generar tablas, porque generaliza dos casos de uso, en este caso son mostrados los diagramas de actividad Generar tablas predefinidas, Generar tablas parciales, Convertir función a tabla, Convertir tabla a función, y Verificación T-T.

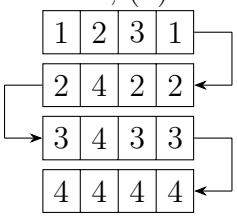
8.1.1. Generar tablas predefinidas

Inicialmente es necesario denotar la estructura de las tablas, la tabla es denotada por conveniencia mediante un arreglo de tamaño 4^n, un ejemplo es mostrado para dos variables en la transformación de la Tabla 8.1 (a) a la Tabla 8.1 (b), cada posición desde el 1 hasta el 16 representa los valores de las variables $(x_1, ..., x_n)$, y el valor de la posición representa el correspondiente valor de verdad $f(x_1, ..., x_n)$ o t, los valores pueden ser identificados como sigue: 1 para 0, 2 para u, 3 para v, y 4 para 1, de la misma manera se hace la transformación para las siguientes filas de la Tabla 8.1 (a).

Tabla 8.1: Manejo de la Tabla. (a) Estandar, (b) Como un arreglo.

x_1 \ x_2	0	u	v	1
0	0	u	v	0
u	u	u	1	u
v	v	1	v	v
1	1	1	1	1

(a)



(b)

Al analizar varias tablas de verdad que reproducen las Funciones lógicas son observados patrones, a estos patrones se les llama valores predefinidos, los cuales pueden ser utilizados para el descarte de gran cantidad de datos, la permutación creada al terminar el algoritmo es la que tiene las tablas de verdad listas para ser filtradas, en la Tabla 8.2 son mostrados valores predefinidos para un conjunto de posiciones en una tabla de verdad denotado de la siguiente manera:

- Para una variable en la Tabla 8.2 (a) son mostrados los valores predefinidos para las posiciones $P_1 = 1$, y $P_1 = 4$, y en la Tabla 8.2 (b) son mostrados los valores predefinidos para las posiciones $(P_2, P_3) = (2, 3)$.
- Con dos variables en la Tabla 8.2 (a) son mostrados los valores predefinidos para las posiciones $P_1 = 1$, $P_1 = 4$, $P_1 = 13$, $P_1 = 16$, en la Tabla 8.2 (b) son mostrados los valores predefinidos para las posiciones $(P_2, P_3) = (2, 3)$, $(P_2, P_3) = (5, 9)$, $(P_2, P_3) = (8, 12)$, y $(P_2, P_3) = (14, 15)$, y en la Tabla 8.2 (c) son mostrados los valores predefinidos para las posiciones $(P_4, P_5, P_6, P_7) = (6, 7, 10, 11)$.

Tabla 8.2: Valores de posiciones para una tabla de verdad. (a) P_1. (b) (P_2, P_3). (c) $(P_4, ..., P_7)$.

P_1
1
4

(a)

P_2	P_3
1	1
4	4
2	3

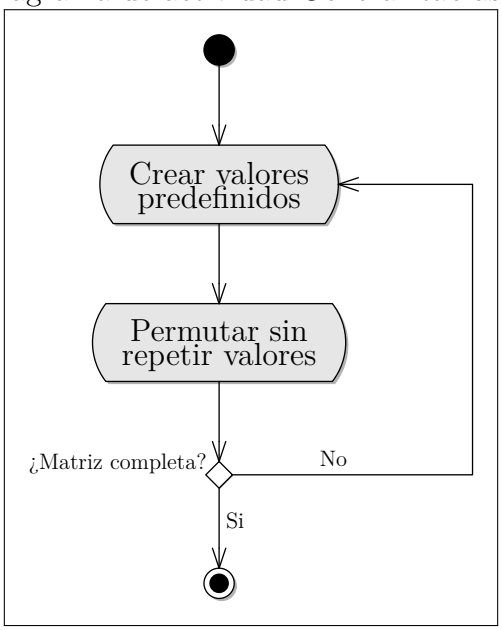
(b)

P_4	P_5	P_6	P_7
1	1	1	1
4	4	4	4
2	1	1	3
2	0	0	3
2	2	3	3
2	3	2	3

(c)

A partir de lo anterior, el diagrama para generar tablas de verdad y su correspondiente algoritmo son presentados a continuación:

Figura 8.1: Diagrama de actividad Generar tablas predefinidas.



1. Los valores predefinidos para la Posición 1 y las posiciones (2,3) son creados.

2. Una permutación sin repetir es realizada para los valores de la posición 1 y las posiciones (2,3).

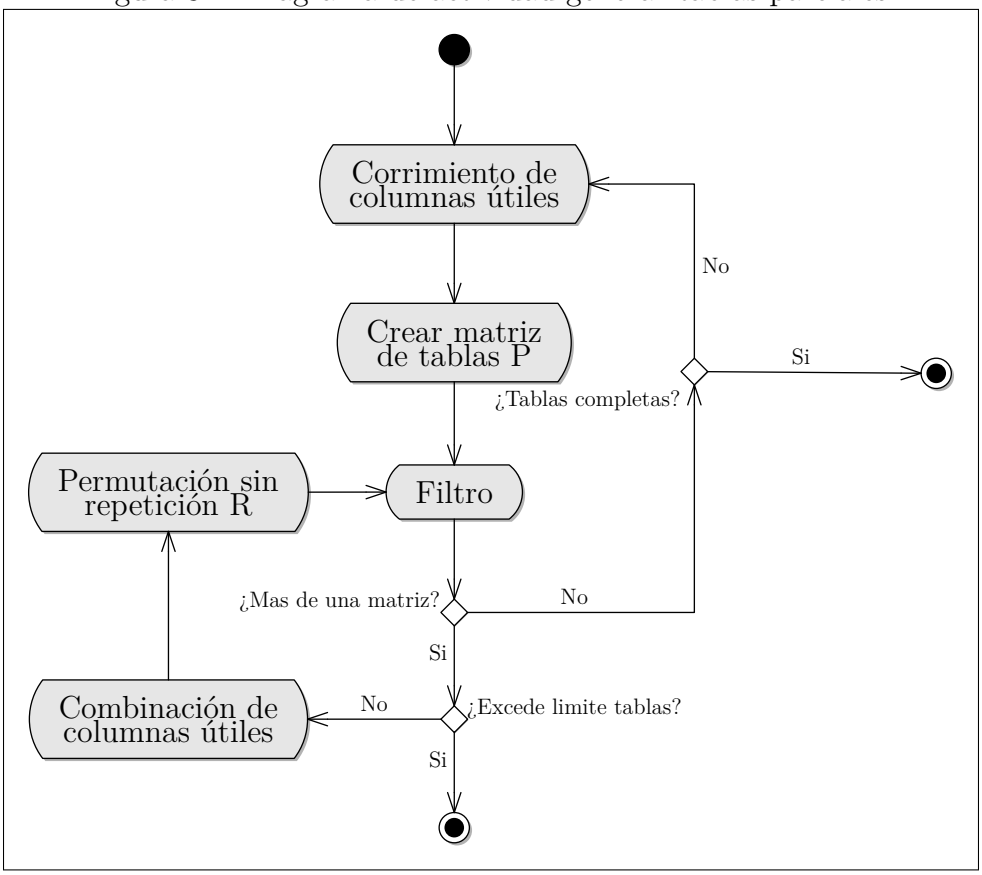
3. Los pasos 1 y 2 son repetidos con valores predefinidos de otras posiciones hasta completar una matriz en la cual las columnas representen los valores $f(x_1, ..., x_n)$ y las filas representen las tablas de verdad.

Nótese que si al aplicar el filtro después del algoritmo no retorna el número de tablas correspondientes dependiendo de la variable, las condiciones no son adecuadas, pero si al aplicar el filtro retorna el número de tablas correspondientes, no significa que las condiciones sean adecuadas, esto es corregido con el diagrama de actividad de Generar Tablas Parciales, una evolución del correspondiente diagrama de actividad.

8.1.2. Generar tablas parciales

El diagrama utilizado para el descarte de gran cantidad de datos depende las matrices P, R antes del Filtro, sus columnas representan los valores $f(x_1, ..., x_n)$ y sus filas representan las tablas de verdad. Las matrices son importantes debido a que no se analiza todas las tablas, sino se escoge y acumulan las columnas a analizar cuando se permutan los datos, las columnas no importantes se llenan con valores 5 o "valores no importa" y se ignora, la matriz R al terminar el algoritmo es la que tiene las tablas de verdad válidas, el diagrama y su correspondiente algoritmo son presentados a continuación:

Figura 8.2: Diagrama de actividad generar tablas parciales.



1. Lo primero por hacer es la creación de la matriz X de 256 filas y 4 columnas, donde todas las posibles permutaciones con repetición de los valores $1, 2, 3, 4$ son almacenados.

2. Un ciclo es creado, el cual cada cuatro columnas son analizadas por iteración, es decir, 4^{n-1} iteraciones. En la primera iteración los pasos del 3 al 8 son dados. Para la segunda iteración en adelante, son dados los pasos del 9 al 12.

3. Dos arreglos más son creados, $rang1$ y $rang2$. Las variables $rang1$ y $rang2$ son arreglos que identificarán cuales columnas (posiciones) de las tablas de verdad serán analizadas. La variable $rang1$ representa el corrimiento de posiciones, y la variable $rang2$ representa la acumulación de posiciones, ambos $rang1$ y $rang2$ tomarán un valor igual a $[1, 2, 3, 4]$.

4. W es creado e igualado a X.

5. Siguiendo esto, la matriz R es creada con el mismo número de filas que W, y 4^n columnas con "valores no importa".

6. Las columnas de valores no importa en R son reemplazadas por los valores de W en las columnas indicadas por la variable $rang2$.

7. La matriz R modificada es enviada como un parámetro a la función Filtro junto con el número de variables n, esta función retornará una matriz con 4^n columnas y tantas filas como combinaciones filtradas sean encontradas, y es asignada a R.

8. Las columnas indicadas por $rang2$ de R son asignadas a R.

9. Ahora $rang1$ tomará el valor de $[4(i-1)+1, ..., 4i]$, y $rang2$ tendrá un valor igual a $[1, 2, ..., 4i]$ donde i es el número de la iteración.

10. Los pasos 5, 6, 7 y 8 son repetidos, pero usando otras variables: P para R, $rang1$ para $rang2$, y X para W.

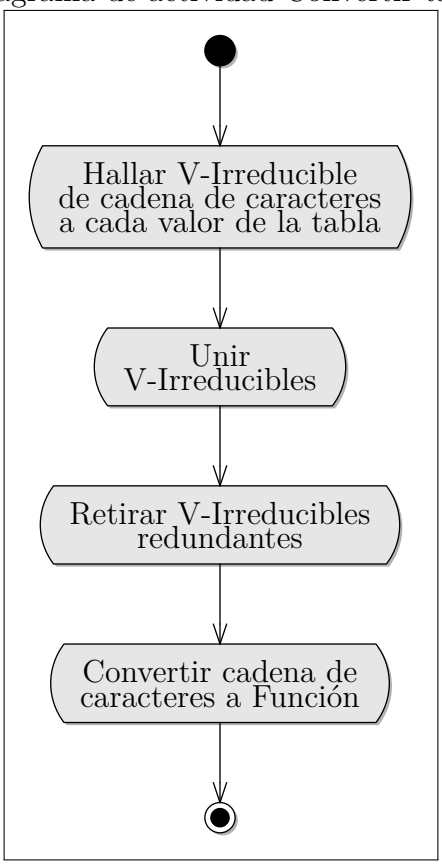
11. Una permutación sin repetición de las matrices resultantes R and P en una matriz modificada W es realizada, si el número de tablas es muy grande se termina el algoritmo.

12. Los pasos 5, 6, 7 y 8 son repetidos otra vez.

8.1.3. Convertir tablas a función

Este diagrama está basado en la formación de FLCR a partir de tablas de verdad, se debe tener en cuenta que la tabla transformada a fórmula no es denotada como arreglo, por ello el manejo del diagrama es matricial, y los valores son 0, 1, u, y v, el diagrama y su correspondiente algoritmo son presentados a continuación:

Figura 8.3: Diagrama de actividad Convertir tabla a función.



1. El $\vee$-irreducible en forma de cadena de caracteres a cada valor de la tabla de verdad es creado, es decir, debe ser obtenido una matriz de $\vee$-irreducibles.

2. Una cadena de caracteres a partir de la unión de $\vee$-irreducibles debe ser obtenido.

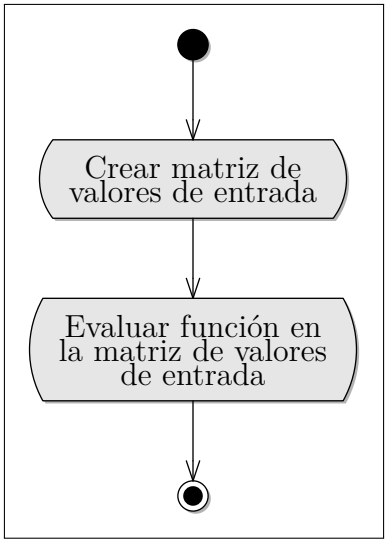
3. A partir de relaciones de absorción descritas en el álgebra de De Morgan, son retirados los $\vee$-irreducibles innecesarios de la cadena de caracteres.

4. Finalmente es convertida la cadena de caracteres a una función, compuesta de subfunciones $\vee$, $\wedge$, y $'$ previamente establecidas.

8.1.4. Convertir función a tabla

La importancia de este diagrama radica en ser el diagrama inverso de Convertir tabla a función, se debe tener en cuenta que la tabla generada por el algoritmo no es denotada como arreglo, y los valores son 0, 1, u, v, el diagrama y su correspondiente algoritmo son descritos a continuación:

Figura 8.4: Diagrama de actividad Convertir función a tabla.



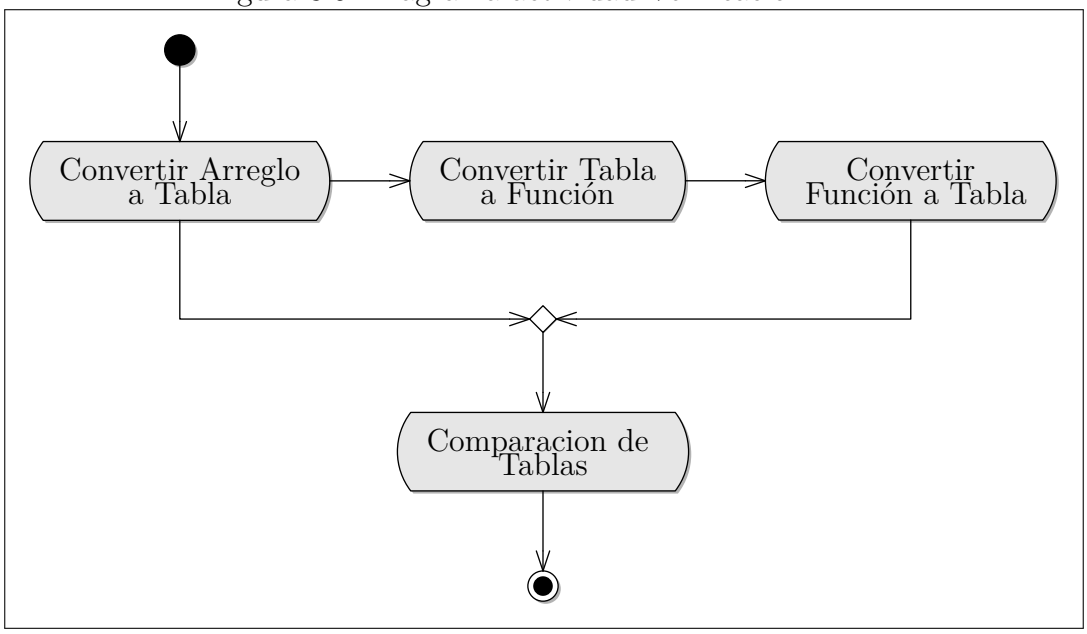
1. Es creada una matriz donde cada posición contenga un arreglo de tantos valores 0, 1, u, v como número de variables.

2. Evaluar la función en cada posición de la matriz creada.

8.1.5. Verificación T-T

Finalmente el diagrama de la verificación T-T o verificación Tabla-Tabla es agregado, el cual se enfoca en la comparación de tablas, el diagrama y su correspondiente algoritmo son descritos a continuación:

Figura 8.5: Diagrama actividad Verificación T-T.

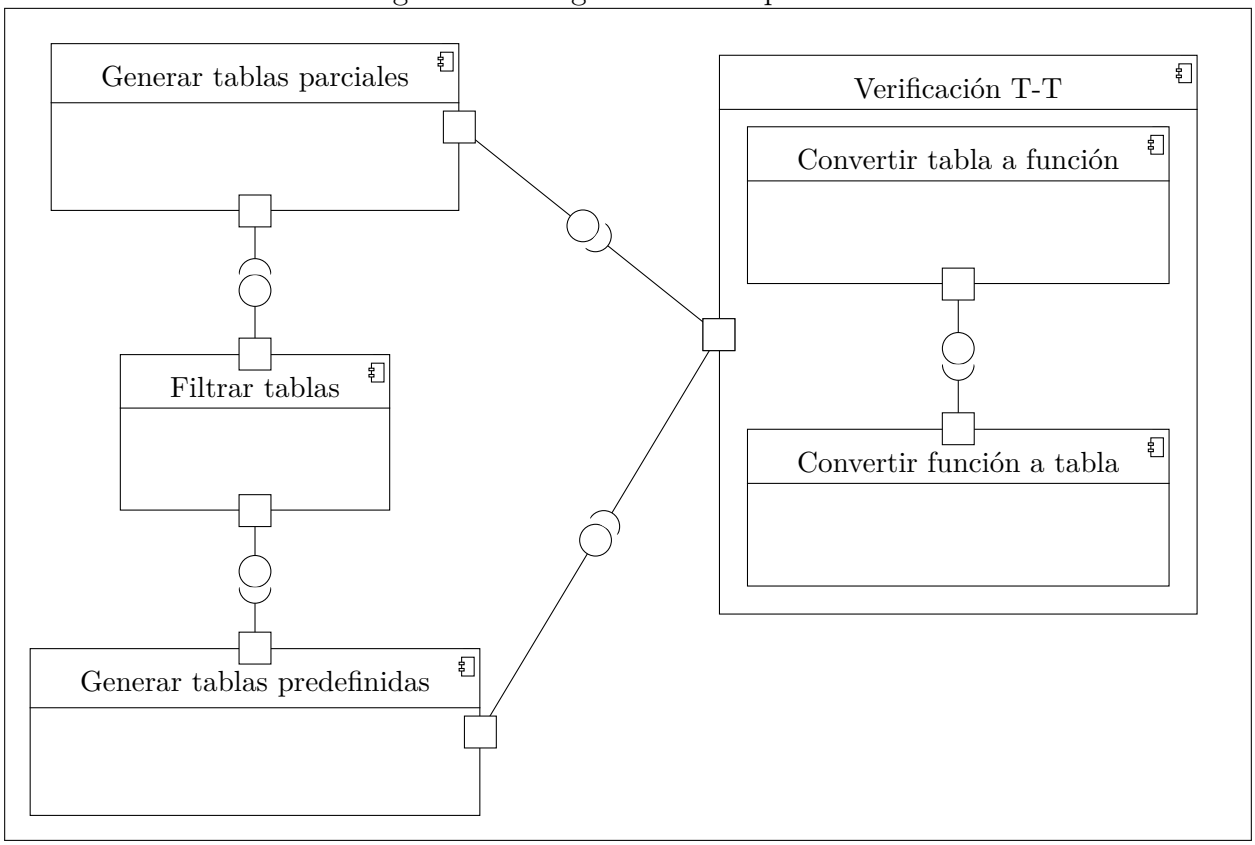


1. El arreglo creado con valores 1, 2, 3, 4 es convertido a matriz con valores 0, u, v, 1.
2. La tabla generada es convertida a función, este paso es descrito anteriormente en el diagrama de actividad Convertir Tabla a Función.
3. La función generada es convertida a tabla, este paso es descrito anteriormente en el diagrama de actividad Convertir Función a Tabla.
4. Se toma las tablas de los pasos 1, 3 y se comparan valor con valor, si todos los valores son iguales la tabla representa la fórmula.

8.2. Diagrama de componentes

Las partes que componen al sistema son mostradas en el diagrama de componentes de la Figura 8.6. Es observado la interconexión entre componentes con otros mediante interfaces a fin de lograr el correcto funcionamiento general, como lo es el caso de Filtrar tablas, Generar tablas predefinidas, Generar tablas parciales, Verificación T-T, Convertir tabla a función, y convertir función a tabla.

Figura 8.6: Diagrama de componentes.



Capítulo 9

CONDICIONES PARA FILTRADO DE TABLAS

Para hacer filtro de las tablas de verdad siguiendo el modelamiento del sistema, es necesario definir las condiciones de filtro, en este capítulo son presentadas las condiciones encontradas en la literatura [13], y las condiciones propuestas.

9.1. Condiciones referenciadas

Las condiciones y definiciones de apoyo mostradas a continuación son propuestas por Soriano en [13], y basadas a partir de lo propuesto por Mukaidono [8] y Walker [10, 11].

9.1.1. Condición 1: Fórmulas bien formadas

Para iniciar con las condiciones de referencia, en esta sección se da la definición de Fórmulas bien formadas, debe denotarse que esta definición es diferente a la definición mostrada en el capítulo del Marco de referencia.

Definición 7 *(Fórmulas bien formadas). Las funciones cuaternarias pueden ser representadas por una fórmula lógica bien formada* f*, compuesta de variables* x_1*,* x_2*, ...,* x_n*), las operaciones lógicas de conjunción* ($\wedge$)*, disyunción* ($\vee$) *negación* $'$*, y las constantes* 0*,* u*,* v*,* 1*.*

9.1.2. Condición 2: Regularidad para la incertidumbre

En esta sección se continua con la definición de Ambigüedad propuesto por Mukaidono [8], la definición análoga de Contradictoriedad y las definiciones de Incertidumbre, y la Regularidad para la incertidumbre.

Definición 8 *(Ambigüedad)* $0 \propto u$*,* $1 \propto u$*,* $i \propto i$*,* $i \in \{0, u, 1\}$*, En la relación* $\propto$*,* 1 *y* 0 *no son comparables. Esta relación puede ser extendida para* $\{0, u, 1\}^n$ *de esta manera: para dos elementos* $\boldsymbol{a} = (a_1, a_2, ..., a_n)$*, y* $\boldsymbol{b} = (b_1, b_2, ..., b_n)$ *de* $\{0, u, 1\}^n$*,* $\boldsymbol{b} \propto \boldsymbol{a}$ *si y solo si* $b_i \propto a_i$ *para*

todos los valores de i. Si $\boldsymbol{b} \propto \boldsymbol{a}$, entonces $\boldsymbol{b}$ es menos ambiguo o igual que $\boldsymbol{a}$.

Definición 9 *(Contradictoriedad) $0\angle v$, $1\angle v$, $i\angle i$, $i \in \{0, v, 1\}$, En la relación $\angle$, 1 y 0 no son comparables. Esta relación puede ser extendida para $\{0, v, 1\}^n$ de esta manera: para dos elementos $\boldsymbol{c} = (c_1, c_2, ..., c_n)$, y $\boldsymbol{b} = (b_1, b_2, ..., b_n)$ de $\{0, v, 1\}^n$, $\boldsymbol{b}\angle\boldsymbol{c}$ si y solo si $b_i \angle c_i$ para todos los valores de i. Si $\boldsymbol{b}\angle\boldsymbol{c}$, entonces $\boldsymbol{b}$ es menos contradictorio o igual que $\boldsymbol{c}$.*

Definición 10 *(Incertidumbre) $0 \ll u$, $0 \ll v$, $1 \ll u$, $1 \ll v$, $i \ll i$, $i \in \{0, u, v, 1\}$, en la relación $\ll$, u, v no son comparables y 1, 0 no son comparables. Si la relación de orden es usada sobre el conjunto $\{0, v, 1\}$, es llamada ambigüedad ($\propto$), y si la relación de orden es usada sobre el conjunto $\{0, v, 1\}$ es llamada contradictoriedad ($\angle$) definidas anteriormente, La relación puede ser extendida a $\{0, u, v, 1\}^n$ y solo son comparables en un mismo sentido de incertidumbre ambigüedad o contradictoriedad.*

Definición 11 *(Regularidad para la incertidumbre). Si $f(\boldsymbol{a}) \in \{0, 1\}$ o $f(\boldsymbol{c}) \in \{0, 1\}$ entonces $f(\boldsymbol{a}) = f(\boldsymbol{c}) = f(\boldsymbol{b})$ para cada $\boldsymbol{a}$, $\boldsymbol{c}$, tal que exclusivamente $\boldsymbol{b} \propto \boldsymbol{a}$ o $\boldsymbol{b}\angle\boldsymbol{c}$ y no importando el tipo de incertidumbre las dos tendrían el mismo valor.*

9.1.3. Condición 3: Monotonicidad para la incertidumbre

A partir de la definición de Incertidumbre, mostrada en la sección anterior, la definición de Monotonicidad para la incertidumbre es mostrada a continuación.

Definición 12 *(Monotonicidad para la incertidumbre) Si $\boldsymbol{b} \propto \boldsymbol{a}$ entonces $f(\boldsymbol{b}) \propto f(\boldsymbol{a})$; de igual manera, si $\boldsymbol{b}\angle\boldsymbol{c}$ entonces $f(\boldsymbol{b})\angle f(\boldsymbol{c})$.*

9.1.4. Condiciones 4 y 5: Incertidumbre combinada

Finalmente, en esta sección se dan las definiciones de Incertidumbre combinada y la función $f(\mathbf{a})$ o $f(\mathbf{c}) \in \{0, 1\}$, e Incertidumbre combinada y la función $f(\mathbf{a})$ o $f(\mathbf{c}) \in \{u, v\}$.

Definición 13 *(Incertidumbre combinada y la función $f(\boldsymbol{a})$ o $f(\boldsymbol{c}) \in \{0, 1\}$). Si $f(\boldsymbol{a}) \in \{0, 1\}$ o $f(\boldsymbol{c}) \in \{0, 1\}$ entonces $f(\boldsymbol{a}) = f(\boldsymbol{c})$ para cada $\boldsymbol{a}$, con al menos un $a_i = u$, $a_j = v$ y $\boldsymbol{c}$ con al menos un $c_i = v$, $c_j = u$ (o viceversa).*

Definición 14 *(Incertidumbre combinada y la función $f(\boldsymbol{a})$ o $f(\boldsymbol{c}) \in \{u, v\}$). Si $f(\boldsymbol{a}) \in \{u, v\}$ o $f(\boldsymbol{c}) \in \{u, v\}$ entonces $f(\boldsymbol{a}) = u$ y $f(\boldsymbol{c}) = v$, o $f(\boldsymbol{a}) = v$ y $f(\boldsymbol{c}) = u$ para cada $\boldsymbol{a}$, con al menos un $a_i = u$, $a_j = v$ y $\boldsymbol{c}$ con al menos un $c_i = v$, $c_j = u$ (o viceversa).*

9.2. Condiciones propuestas

Conforme al conocimiento acerca de la formación sobre las Funciones Lógicas Cuaternarias Regulares, son establecidas algunas condiciones las cuales una tabla de verdad puede ser representada por una fórmula en $\mathbb{L}_{\mathbb{D}}$.

Lo primero es crear una tabla eligiendo en el retículo L_D, algunos elementos o Formas Normales Disyuntivas (FND) que sean $\vee$-irreducibles, con los t, y los valores de las variables $x_1, x_2, ..., x_n \in V$ adecuados.

Las primeras tres condiciones son basadas de [26] y del Teorema 1 anteriormente mencionado. El procedimiento para generar las condiciones es el siguiente:

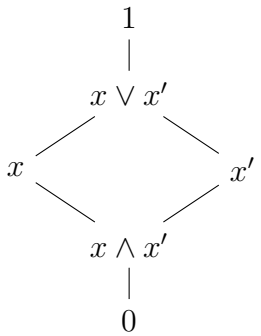
- Con los elementos elegidos generar el poset L_j
- Después, generar el poset con los valores de las variables o vectores para cada fila en la tabla, teniendo la misma estructura que L_j.
- Observando el poset final de vectores, crear la definición de orden y las condiciones que la usan.

Las dos condiciones restantes surgen al agrupar o al observar igualdades entres las formas normales o los valores de la tabla.

9.2.1. Condición 1: Regularidad

De acuerdo a lo anterior, el retículo L_D será usado para una variable $x \in V$, mostrado en la Figura 9.1 [27].

Figura 9.1: Retículo L_D para una variable x.



Los valores de las variables y los t se eligen de manera que $x = \{0, u, 1\}$, y $t = 1$, mostrados en la Tabla 9.1, para generar el correspondiente poset L_j, y el poset final mostrados en la Figura 9.2.

Tabla 9.1: Formas Normales elegidas de L_D para x.

x	t	FND
0	1	x'
1	1	x
u	1	1

Figura 9.2: Posets generados de L_D para x. (a) poset de $\vee$-irreducibles L_j, (b) poset de valores.

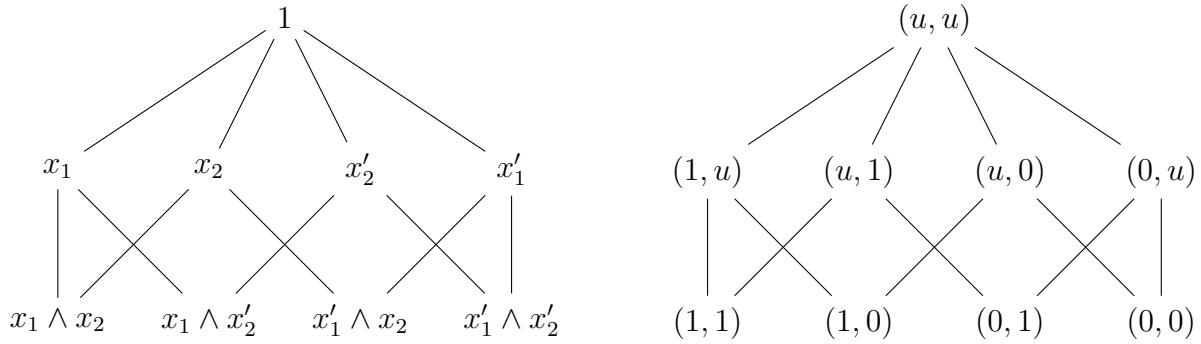
(a) (b)

En la Figura 9.2 (b), un nuevo orden es creado. Para dos variables x_1 y x_2, los valores son elegidos de manera que $x_1, x_2 = \{0, u, 1\}$, y $t = 1$, esto es mostrado en la Tabla 9.2, y los posets son mostrados en la Figura 9.3.

Tabla 9.2: Formas Normales elegidas de L_D para x_1, x_2.

x_1	x_2	t	FND
u	u	1	1
1	u	1	x_1
0	u	1	x_1'
u	1	1	x_2
u	0	1	x_2'
1	0	1	$x_1 \wedge x_2'$
0	1	1	$x_1' \wedge x_2$
1	1	1	$x_1 \wedge x_2$
0	0	1	$x_1' \wedge x_2'$

Figura 9.3: Posets generados de L_D para x_1, x_2. (a) poset de $\vee$-irreducibles L_j, (b) poset de valores.



(a) (b)

En la Figura 9.3 (b), el orden es generalizado para varios valores y esto pasa si el valor de t es igual a 1, análogamente, si la Forma Normal Conjuntiva es usada, t debe ser igual a 0. Ambos, el orden generalizado (ambigüedad) y la condición pertenecen a Kleene y Mukaidono [3, 8], el orden está definido anteriormente, la condición es definida como sigue:

Definición 15 *(Regularidad) Si* $f(\boldsymbol{a}) \in \{0,1\}$, $f \in L_D$ *para* $x_1, x_2, ..., x_n$, *entonces* $f(\boldsymbol{b}) = f(\boldsymbol{a})$ *para cada* $\boldsymbol{b}$ *tal que* $\boldsymbol{b} \propto \boldsymbol{a}$.

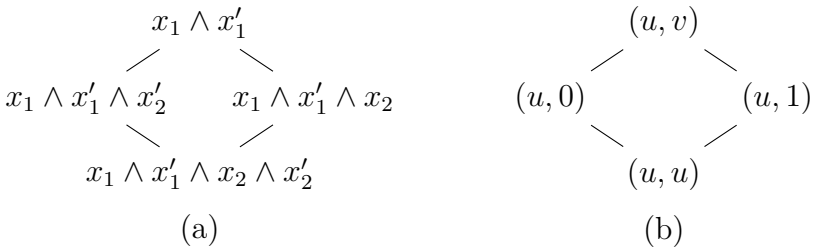
9.2.2. Condición 2: Reducibilidad

Esta condición aparece cuando una forma de organización diferente a la ambigüedad de Mukaidono es estudiada. Para dos variables x_1 y x_2 los valores son elegidos de manera que $x_1 = u$, $x_2 \in \{0, u, v, 1\}$, y $t = u$, esto es mostrado en la Tabla 9.3, y los posets son mostrados en la Figura 9.4.

Tabla 9.3: Formas Normales elegidas de L_D para x_1, x_2.

x_1	x_2	t	FND
u	v	u	$x_1 \wedge x_1'$
u	0	u	$x_1 \wedge x_1' \wedge x_2'$
u	1	u	$x_1 \wedge x_1' \wedge x_2$
u	u	u	$x_1 \wedge x_1' \wedge x_2 \wedge x_2'$

Figura 9.4: Posets generados de L_D para x_1, x_2. (a) poset de $\vee$-irreducibles L_j, (b) poset de valores.



En la Figura 9.4 (b) se aprecia que si la segunda posición de cada vector es tomada, un nuevo orden es creado, esto puede ser generalizado para varios valores, por lo tanto el orden y la condición son definidos como sigue:

Definición 16 *(Lateralidad)* $0 \prec v$, $1 \prec v$, $u \prec 0$, $u \prec 1$, $i \prec i$, $i \in \{0, u, v, 1\}$, *En la relación* $\prec$, 1 *y* 0 *no son comparables. Esta relación puede ser extendida para* $\{0, u, v, 1\}^n$ *de esta manera: para dos elementos* $\boldsymbol{a} = (a_1, a_2, ..., a_n)$, *y* $\boldsymbol{b} = (b_1, b_2, ..., b_n)$ *de* $\{0, u, v, 1\}^n$, $\boldsymbol{b} \prec \boldsymbol{a}$ *si y solo si* $b_i \prec a_i$ *para todos los valores de* i. *Si* $\boldsymbol{b} \prec \boldsymbol{a}$, *entonces* $\boldsymbol{b}$ *es menos lateral o igual que* $\boldsymbol{a}$.

Definición 17 *(Reducibilidad) Si* $f(\boldsymbol{a}) = u$, $f \in L_D$ *para* x_1, x_2, ..., x_n *y existe al menos un* $a_i = u$, *entonces* $f(\boldsymbol{b}) = f(\boldsymbol{a})$ *para cada* $\boldsymbol{b}$ *tal que* $\boldsymbol{b} \prec \boldsymbol{a}$.

9.2.3. Condición 3: Inversión

La siguiente condición surge de elegir los valores tales que $x_1 = u$, $x_2 \in \{0, u, 1\}$, y $t \in \{1, u\}$, en adición al caso $x_1 = u$, $x_2 = v$ y $t = 1$ para dos variables x_1 y x_2, la Tabla 9.4 es generada.

Tabla 9.4: Formas Normales elegidas de L_D para x_1, x_2.

x_1	x_2	t	FND
u	v	1	$x_1 \wedge x_1'$ y $x_2 \wedge x_2'$
u	0	1	x_2'
u	1	1	x_2
u	u	1	1
u	0	u	$x_1 \wedge x_1' \wedge x_2'$
u	1	u	$x_1 \wedge x_1' \wedge x_2$
u	u	u	$x_1 \wedge x_1' \wedge x_2 \wedge x_2'$

Como hay dos $\vee$-irreducibles para $x_1 = u$ y $x_2 = v$ cuando $t = 1$, en lugar de hacer la disyunción, los $\vee$-irreducibles serán tratados como si ellos fueran dos formas normales separadas, por lo tanto dos posets son generados: L_{j1} y L_{j2}, con sus posets de valores, esto es mostrado en la Figura 9.5 y la Figura 9.6.

Figura 9.5: Posets generados de L_D para x_1, x_2. (a) poset de $\vee$-irreducibles L_{j1}, (b) poset de valores.

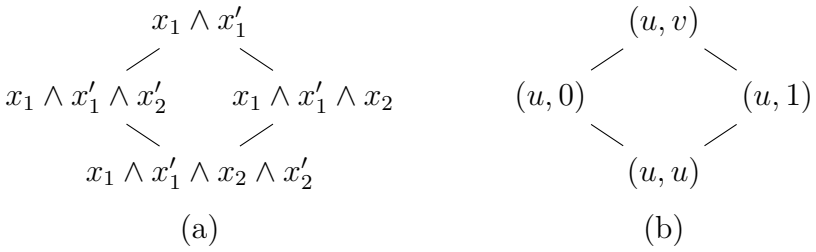
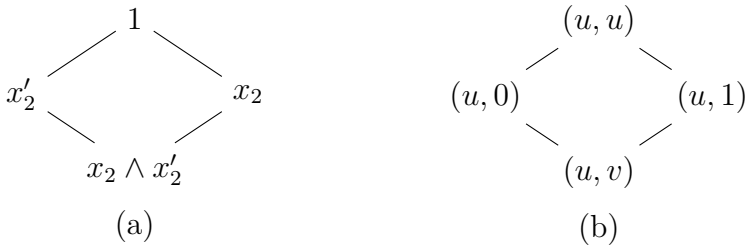


Figura 9.6: Posets generados de L_D para x_2. (a) poset de $\vee$-irreducibles L_{j2} (b) poset de valores.



Debido a la similaridad entre la Figura 9.4 y la Figura 9.5 por el orden de lateralidad, para crear la condición se mantiene el orden de lateralidad (o invierte) en los posets de la Figura 9.6. Análogamente, con la Forma Normal Conjuntiva, los valores tales que $t = 0$ son tomados y los mismos resultados son obtenidos.

Definición 18 *(Inversión) Si $f(\boldsymbol{a}) \in \{0, 1\}$, $f \in L_D$ para x_1, x_2, ..., x_n, y existe al menos un $a_i = u$, entonces $f(\boldsymbol{b}) = f(\boldsymbol{a})$ or $f(\boldsymbol{b}) = u$ para cada $\boldsymbol{b}$ tal que $\boldsymbol{b} \prec \boldsymbol{a}$.*

9.2.4. Condición 4: Repetición

En este caso los posets no son creados, pero, una igualdad es encontrada, comenzando con elegir los valores de las variables y los t tales que $x = \{u, v\}$, $t = 1$, con $x = u$, $t = u$ y $x = v$, $t = v$, esto es mostrado en la Tabla 9.5.

Tabla 9.5: Formas Normales elegidas en L_D para x.

x	t	FND
u	1	1
v	1	1
u	u	$x \wedge x'$
v	v	$x \wedge x'$

Las formas normales son repetidas si u es cambiado por v y viceversa, pero para valores iguales a 1 o 0 lo mismo no pasa, por lo tanto otra condición es establecida.

Definición 19 *(Intercambio) Sea $R(x)$ una función a trozos mostrada abajo en la ecuación 9.1:*

$$R(x) = \begin{cases} 1 & si \quad x = 1 \\ 0 & si \quad x = 0 \\ u & si \quad x = v \\ v & si \quad x = u \end{cases} \tag{9.1}$$

Esto puede ser generalizado como sigue: para cada $\boldsymbol{a} = (a_1, a_2, ..., a_n) \in \{0, u, v, 1\}^n$ tal que $R(\boldsymbol{a}) = (R(a_1), R(a_2), ..., R(a_n))$.

Definición 20 *(Repetición) Si $f \in L_D$ para x_1, x_2, ..., x_n, entonces $f(R(\boldsymbol{a})) = R(f(\boldsymbol{a}))$.*

9.2.5. Condición 5: Restricción

Esta condición surge al elegir todos los valores de las variables y los t posibles de la tabla y agrupar cuales valores y cuales t pertenecen a los conjuntos $\{0, 1\}^n$, $\{0, u, 1\}^n$, $\{0, v, 1\}^n$, y $\{0, u, v, 1\}^n$ [10, 11, 12, 20].

Definición 21 *(Restricción) Para $f \in L_D$ para x_1, x_2, ..., x_n, y $\boldsymbol{a} = (a_1, a_2, ..., a_n)$:*

- *Si $\boldsymbol{a} \in \{0, 1\}^n$, entonces $f(\boldsymbol{a}) \in \{0, 1\}$.*
- *Si $\boldsymbol{a} \in \{0, u, 1\}^n$, entonces $f(\boldsymbol{a}) \in \{0, u, 1\}$.*
- *Si $\boldsymbol{a} \in \{0, v, 1\}^n$, entonces $f(\boldsymbol{a}) \in \{0, v, 1\}$.*
- *Si $\boldsymbol{a} \in \{0, u, v, 1\}^n$, entonces $f(\boldsymbol{a}) \in \{0, u, v, 1\}$.*

9.2.6. Ejemplo

De acuerdo con lo anterior, un ejemplo en el cual se usan las condiciones para verificar si la Tabla 9.6 en el álgebra de De Morgan puede ser representa por una función lógica cuaternaria regular es presentado a continuación:

Tabla 9.6: Tabla de ejemplo f_1

x_1 \ x_2	0	u	v	1
0	1	u	1	1
u	1	u	v	u
v	1	u	v	v
1	u	u	v	1

Al realizar la verificación de las condiciones en la Tabla 9.6 no satisface las siguientes condiciones:

- Regularidad: $f_1(1,0) \neq f_1(u,0)$, siendo $(1,0) \propto (u,0)$ y $f_1(u,0) = 1 \in \{0,1\}$.
- Repetición: $f_1(R(1,0)) \neq R(f_1(1,0))$.
- Restricción: $f_1(1,0) = u \notin \{0,1\}$.

Por el incumplimiento de las condiciones anteriormente mostradas, f_1 no puede ser representada por una Función Lógica Cuaternaria Regular.

Capítulo 10

RESULTADOS

En este capítulo son mostrados resultados a partir de la aplicación de las condiciones referenciadas por la literatura y las condiciones propuestas a los algoritmos correspondientes.

10.1. Condiciones referenciadas

En esta sección es presentada la aplicación de los algoritmos Generar Tablas predefinidas y Verificación T-T, a las condiciones encontradas en la literatura.

10.1.1. Generar Tablas predefinidas

Los resultados obtenidos del algoritmo Generar tablas predefinidas y posteriormente Filtrar tablas para las tablas de verdad de 1 variable con 6 tablas y 2 variables con 330 tablas muestra que para una variable coincide con el segundo número de Dedeking [12], pero para 2 variables no, esto implica que se debe verificar las tablas para 1, y 2 variables pero no es necesario probar las condiciones usando el algoritmo Generar tablas parciales, ni aumentar el número de variables a 3.

10.1.2. Verificación T-T

De las tablas generadas por el algoritmo Generar tablas predefinidas para 2 variables se observó que de las 330 tablas de verdad el número de tablas que quedan es 168, y para 1 variable de las 6 tablas sigue quedando las 6 tablas, lo que implica que para dos variables sobran tablas, y posiblemente falten condiciones para que las tablas de verdad representen la fórmula.

10.2. Condiciones propuestas

Continuando con los resultados, en esta sección es presentada la aplicación de los algoritmos Generar Tablas predefinidas, Generar Tablas parciales y Verificación T-T, a las condiciones propuestas.

10.2.1. Tablas predefinidas

Los resultados obtenidos del algoritmo Generar tablas predefinidas y posteriormente Filtrar tablas para las tablas de verdad de 1 variable con 6 tablas y 2 variables con 168 tablas muestra que coincide con el $2n$ número de Dedeking [12], dando un punto de partida para el algoritmo Generar tablas parciales.

10.2.2. Tablas parciales

Los resultados que fueron obtenidos del algoritmo serán mostrados en las Tablas 10.1, 10.2, 10.3, para las tablas de verdad de 1, 2 y 3 variables respectivamente, donde son representados el número de filas en las matrices **P**, **R** (Número de tablas), y **W** (Término si excede límite) por cada iteración, las tablas son mostradas a continuación:

Tabla 10.1: Número de filas de matrices para 1 variable.

Iteración	**P**	**W**	**R**
1	0	256	6

Tabla 10.2: Número de filas de matrices para 2 variables.

Iteración	**P**	**W**	**R**
1	0	256	6
2	22	132	62
3	108	6 696	50
4	6	300	168

Tabla 10.3: Número de filas de matrices para 3 variables.

Iteración	**P**	**W**	**R**
1	0	256	6
2	22	132	62
3	108	6 696	50
4	6	300	168
5	22	3 696	1 698
6	22	37 356	3 458
7	36	124 488	47 878
8	22	1 053 316	275 554
9	108	29 759 832	237 950
10	36	8 566 200	181 740
11	108	19 627 920	181 740
12	108	19 627 920	160 948
13	6	965 688	592 910
14	22	13 044 020	2 999 342
15	108	323 928 936	2 687 908
16	6	16 127 448	7 828 354

Como puede observarse, el número de filas en **R** al finalizar la última iteración en las tablas anteriores, coincide con el $2n$ número de Dedeking [12], dando un buen punto de partida para hacer la Verificación T-T.

10.2.3. Verificación T-T

Como herramienta computacional para la gestión de matrices y arrays, la computadora CECAD de la Universidad Distrital que tiene 56 núcleos de procesamiento se ha utilizado, reduciendo el tiempo paralelizando el algoritmo Verificación T-T obteniendo los resultados mostrados en la Figura 10.1.

Figura 10.1: Resultados verificaciones T-T satisfactorias en la CECAD.

```
>> Importar
>> exit
[cecad@sabioII-5 Algebras]$ nano Walker.m
[cecad@sabioII-5 Algebras]$ clear
[cecad@sabioII-5 Algebras]$ matlab
MATLAB is selecting SOFTWARE OPENGL rendering.

                          < M A T L A B (R) >
                Copyright 1984-2017 The MathWorks, Inc.
                 R2017a (9.2.0.556344) 64-bit (glnxa64)
                            March 27, 2017


To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>> Importar
>> Walker
Starting parallel pool (parpool) using the 'local' profile ...
connected to 28 workers.

ans =

     7828354

>> IdleTimeout has been reached.
Parallel pool using the 'local' profile is shutting down.
>>
```

De acuerdo con lo anterior, las tablas generadas con los algoritmos Generar tablas predefinidas, y generar tablas parciales para 1, 2, y 3 variables se observó que todas las tablas de verdad se conservan, lo que implica que las condiciones son necesarias y suficientes para que las tablas de verdad en el álgebra de De Morgan puedan ser representadas por una fórmula.

Capítulo 11

CONCLUSIONES

Un prototipo de software es implementado sobre el cual es posible probar condiciones para que una tabla de verdad en $\mathbb{L}_{\mathbb{D}}$ pueda ser representada mediante una función lógica.

Es aplicado un nuevo algoritmo de análisis para una gran cantidad de datos, porque no es necesario generar 4^{4^2} tablas de verdad para dos variables y 4^{4^3} para tres variables, optimizando el proceso con un método alternativo mostrado en el generador de tablas parciales, el generador de tablas predefinidas y la Verificación T-T.

En el trabajo realizado por Mukaidono en [8], las condiciones necesarias y suficientes son establecidas para funciones lógicas ternarias regulares, pero estas condiciones no son adecuadas para funciones lógicas cuaternarias regulares. Sin embargo, Gehrke, Carol Walker y Elbert Walker [11] mediante un homomorfismo propuesto logra encontrar 168 funciones regulares en $\mathbb{L}_{\mathbb{D}}$ para dos variables.

Lo anterior permite observar que en las 5 condiciones propuestas en el trabajo desarrollado hay una relación entre los números pares de Dedeking [12] y el número de funciones lógicas cuaternarias regulares, mostrando que: para una variable hay 6 funciones regulares, para dos variables hay 168 y para 3 variables hay 7828354.

En este documento no solo son propuestas condiciones necesarias y suficientes para que una función sea regular en $\mathbb{L}_{\mathbb{D}}$ de una, dos y tres variables, sino que las condiciones pueden ser generalizadas para ser condiciones necesarias en $\mathbb{L}_{\mathbb{D}}$ de n variables.

Capítulo 12

TRABAJO FUTURO

Son dadas varias propuestas para un posible trabajo futuro, que puedan ampliar horizontes al trabajo desarrollado, las propuestas son presentadas a continuación:

- Desarrollar prototipos de software para probar condiciones que deben cumplir funciones lógicas cuaternarias regulares utilizando otras metodologías ágiles conocidas como Scrum, XP, etc.
- Ampliar los métodos de análisis de grandes cantidades de datos para descartar tablas de verdad que no puedan ser representadas por funciones lógicas basados en el manejo de retículos y en la observación de tablas de verdad.
- Desarrollar prototipos de software para probar condiciones que deben cumplir otro tipo de funciones lógicas no booleanas buscando otros enfoques, por ejemplo para las funciones lógicas ternarias regulares [8].
- Proponer condiciones para toda lógica proposicional creada a partir de un álgebra finita basada en retículos en la cual su función lógica pueda ser representada por su correspondiente tabla de verdad [9].
- A partir de las condiciones propuestas, desarrollar un concresor basado en las funciones lógicas cuaternarias regulares usando como base el CBR [1].

Capítulo 13

ANEXO

Los trabajos acerca del álgebra de De Morgan, la lógica proposicional, y las funciones lógicas cuaternarias regulares tienen un fundamento en las relaciones de los conjuntos y la teoría de retículos, presentados a continuación.

13.1. Relaciones

Para la definición de los retículos, un buen punto de partida es definir distintos tipos de relaciones entre conjuntos, y también debe ser denotada la diferencia entre una relación de equivalencia y una relación de orden, las cuales se verán en esta sección.

Definición 22 *(Relación de equivalencia) Una relación binaria (=) definida sobre el conjunto S es una relación de equivalencia sobre el conjunto S si satisface las siguientes condiciones para todos los $a, b, c \in S$* [24, 25].

$$\text{(Reflexibilidad) } a = a \tag{13.1}$$

$$\text{(Simetría) } Si\ a = b,\ entonces\ b = a \tag{13.2}$$

$$\text{(Transitividad) } a = b\ y\ b = c, entonces\ a = c \tag{13.3}$$

Definición 23 *(Clase de equivalencia) Sean un conjunto S y una relación (=) de equivalencia sobre S, si $a \in S$, los elementos $y \in S$ que verifican $a = y$ constituyen un subconjunto, $[a]$, de S, llamado clase de equivalencia* [24]*, Así:*

$$[a] = \{y;\ y \in S,\ y = a\} \tag{13.4}$$

Definición 24 *(Orden parcial) Una relación binaria ($\leq$) definida sobre el conjunto S es un orden parcial sobre el conjunto S si satisface las siguientes condiciones para todos los $a, b, c \in S$* [25].

$$\text{(Reflexibilidad) } a \leq a \tag{13.5}$$

$$\text{(Antisimetría) } a \leq b\ y\ b \leq a, entonces\ a = b \tag{13.6}$$

$$\text{(Transitividad) } a \leq b\ y\ b \leq c, entonces\ a \leq c \tag{13.7}$$

Un conjunto no vacío con un orden parcial es llamado un conjunto parcialmente ordenado, o más simplificando poset.

13.2. Retículos

En esta sección se continua con la definición de retículos, y los distintos tipos de retículos, es denotada la diferencia entre distintos tipos de retículos, sea por la agregación o eliminación de operaciones y propiedades.

Definición 25 *(Retículo) Un poset L es un retículo si para cada $a, b \in L$, ambos $sup\{a, b\}$ e $inf\{a, b\}$ existen en L [25].*

Definición 26 *(Retículo) Un conjunto no vacío L junto a dos operaciones binarias $\wedge$ y $\vee$ sobre L es llamado retículo con notación $(L, \wedge, \vee)$ si satisface las siguientes propiedades para todo $a, b, c \in L$:*

Tabla 13.1: Propiedades de los elementos de un retículo.

Nombre	**Propiedad**
Idempotencia	$a \vee a = a; \quad a \wedge a = a$
Conmutativa	$a \vee b = b \vee a$ $a \wedge b = b \wedge a$
Asociativa	$(a \vee b) \vee c = a \vee (b \vee c)$ $(a \wedge b) \wedge c = a \wedge (b \wedge c)$
Absorción	$a \vee (a \wedge b) = a$ $a \wedge (a \vee b) = a$

Si L es un retículo por la primera definición, entonces se definen las operaciones $\vee$ y $\wedge$ por $a \vee b = sup\{a, b\}$ y $a \wedge b = inf\{a, b\}$ [25].

Definición 27 *(Retículo distributivo) Es un retículo $(L, \wedge, \vee)$ que satisface la siguiente propiedad para todos los $a, b, c \in L$ [25]:*

$$\text{(Distributiva)} \quad (a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c); \quad (a \wedge b) \vee c = (a \vee c) \wedge (b \vee c) \tag{13.8}$$

Definición 28 *(Retículo acotado) Un álgebra $(L, \wedge, \vee, 0, 1)$ con dos operaciones binarias y dos operaciones nularias es un retículo acotado si $(L, \wedge, \vee)$ es un retículo y se satisface la siguiente propiedad para todos los $a, b, c \in L$ [25]:*

$$\text{(Absorción por 1 y 0)} \quad a \vee 1 = 1; \; a \wedge 0 = 0 \tag{13.9}$$

Bibliografía

[1] Espitia, H.E. y Soriano, J.J. Sistema de inferencia difusa basado en relaciones Booleanas. En: Ingeniería, Vol. 15, No. 2, pp. 52-66, 2010.

[2] Hachtel G.D. and Somenzi F., Logic Synthesis and Verification Algorithms, Kluwer Academic Publishers, Dordrecht, 1996.

[3] Kleene S.C. Introduction to metamathematics, vol. 1 of Bibliotheca Mathematica. A Series of Monographs on Pure and Applied Mathematics, North-Holland Publishing Co., Amsterdam, 1952.

[4] Precup R., Hellendoorn Hans, A survey on industrial applications of fuzzy control, Computers in Industry Vol. 62, pp. 213–226, 2011.

[5] Klir G.J. and Yuan B., Fuzzy sets and fuzzy logic: theory and applications, Prentice Hall PTR, New Jersey, 1995.

[6] Gehrke M, Walker CL, Walker EA, Some Comments on Interval Valued Fuzzy Sets, International Journal of Intelligent Systems, Vol. 11, pp. 751–759, 1996.

[7] Salazar O., Soriano J.J. Método de simplificación de fórmulas por medio de álgebras finitas, 2018.

[8] Mukaidono M. Regular ternary logic functions, ternary logic functions suitable for treating ambiguity. IEEE transactions on computers, Vol. c-35, No. 2, pp 179-183, February 1986.

[9] Gehrke M, Walker CL, Walker EA. Mathematical setting for fuzzy logic, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 5, pp. 223–238, 1997.

[10] Gehrke M, Walker CL, Walker EA, Normal forms and truth tables for fuzzy logics, Fuzzy sets and systems Vol. 138, pp 25-51, 2003.

[11] Gehrke M, Walker CL, Walker EA, Normal forms and truth tables for interval-valued fuzzy logic, in Joint 9th IFSA World Congress and 20th NAFIPS International Conference, Vol. 5, Vancouver, British Columbia, Canada, pp. 1327-1331, 2001.

[12] Berman J, Köhler P, Cardinalities of finite distributive lattices, Mitt. Math. Sem. Giessen Vol. 121, pp 103-124, 1976.

[13] Soriano J.J. Álgebra Abstracta Aplicada a Ingeniería, Editorial UD Universidad Distrital Francisco José de Caldas, 2018.

[14] Pressman R.R. Ingeniería de software, un enfoque práctico, Séptima edición, 2010.

[15] PU, Flujo de trabajo del proceso unificado 2020. [Online]. Disponible: https://www.researchgate.net/figure/Flujos-de-trabajo-del-proceso-unificado_fig1_ 279751761. [Accedido: 01-Jun-2020].

[16] PUA, proceso unificado ágil 2020. [Online]. Disponible: http://www.ambysoft.com/ unifiedprocess/agileUP.html. [Accedido: 09-Jun-2020].

[17] Cockburn, A., Writing Effective Use-Cases, Addison-Wesley, 2001.

[18] Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M., Pattern-Oriented Software Architecture A System of Patterns, Wiley Editorial, 2001.

[19] Fidytek R., Mostowski A.W., Somla R., Szepietowski A. Algorithms counting monotone Boolean functions. Information Processing Letters Vol. 79, pp 203-209, 2001.

[20] Berman J. and Mukaidono M., Enumerating fuzzy switching functions and free Kleene algebras, Corp. & Maths. with Appls., Vol. 10, No. I, pp 25-35, 1984.

[21] Fernandes, T. A Kleene Algebra Framework for Data Flow Analysis, Australian Software Engineering Conference (ASWEC'07). 2007.

[22] Kozen D. Kleene Algebra with Tests, ACM Transactions on Programming Languages and SystemsMay, Vol. 19, pp. 427-443, 1997.

[23] Pous D. Symbolic Algorithms for Language Equivalence and Kleene Algebra with Tests, ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Mumbai, India, Vol. 42, pp. 357-368, 2015.

[24] Ayres F. Jr, Álgebra Moderna, Schaum's outline series, pp 1-17, 2003.

[25] Burris S., Sankappanavar H.P. A Course in Universal Algebra, vol. 78 of Graduate Texts in Mathematics, Springer-Verlag, New York, 2012.

[26] Harding J., Walker C.L., and Walker E.A. Partial Orders on the Truth Value Algebra of Type-2 Fuzzy Sets, IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS), pp. 1-3, 2013.

[27] Font J.M.& Moussavi M., Note on a sixvalued extension of three-valued logic, Journal of Applied Non-Classical Logics, Vol. 3, pp. 173-187, 1993.

[28] Mukaidono M. Algebraic structures of interval truth values in fuzzy logic, in Proceedings of the Sixth IEEE International Conference On Fuzzy Systems, vol. 3, Barcelona, Spain, pp. 699–705, 1997.